AD-A074 552 NAVAL UNDERWATER SYSTEMS CENTER NEW LONDON CT NEW LO--ETC F/G 5/2
PERFORMANCE AND TIMELINESS IN A DATABASE.(U)
JUL 79 L A DENOIA
UNCLASSIFIED NUSC-TR-6099 NL

1 OF 3
AD
A074552

1.0 2.8 2.5
3.2 2.2
3.6
4.0 2.0
1.1
1.8
1.25 1.4 1.6

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

NUSC Technical Report 6099

LEVEL

Performance and Timeliness In a Database

Lynn A DeNoia
Special Projects Department

3 July 1979

NUSC
Naval Underwater Systems Center
Newport, Rhode Island • New London, Connecticut

Approved for public release; distribution unlimited.

PREFACE

This research was supported in part under NUSC Project No. A70240, "Computer Technology Review," Principal Investigator: Dr. J. C. Lamb (Code 314), project element No. 62766N, subproject/task No. ZF61-112-001, "Independent Exploratory Development - Target Surveillance," Project Manager: J. H. Probus, Naval Material Command (Code MAT 08T1).

This report was initially prepared for submittal to the Department of Computer Science, Brown University, in partial fulfillment of the requirements for the Degree of Doctor of Philosophy. It was reviewed for NUSC by Dr. J. C. Lamb.

REVIEWED AND APPROVED: 3 July 1979

W A VonWinkle

W. A. VonWinkle
Associate Technical Director
for Technology

The author of this report is located at the New London
Laboratory, Naval Underwater Systems Center,
New London, Connecticut 06320.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR 6099	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) PERFORMANCE AND TIMELINESS IN A DATABASE,		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Lynn A./DeNoia		8. CONTRACT OR GRANT NUMBER(s) (12) 231
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Underwater Systems Center New London Laboratory New London, CT 06320		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS A70240
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Material Command (Code MAT 08T1) Washington, DC 20360		12. REPORT DATE (11) 3 July [redacted]79
		13. NUMBER OF PAGES 227
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (14) NUSC-TR-6Ø99		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for public release; distribution unlimited. (9) Technical rept.,

18. SUPPLEMENTARY NOTES

(16) F61112 (17) ZF61112001

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A methodology is presented for evaluating the system cost/performance of alternative approaches to distributed database management. For each type of database transaction, the management schemes are analyzed to identify the specific control paths and data flow requirements. Then the control and data flow information is used to develop a queuing network model of the entire system. Specific cost/performance analyses can be made when assumptions about particular operating characteristics (such as communication delays, processor power, and disk rates) are incorporated into the model. (over)

DD FORM 1 JAN 73 1473

Over

405 918

20. (Cont'd)

Average system response time and average network message traffic are computed for four management approaches: centralized, a master/slave scheme, a synchronized scheme, and a new scheme called delayed synchronization. The new scheme is based on daily operation without synchronizing updates, supported by nightly merging to produce identical data copies throughout the system. Timeliness information is associated with every individual data item and users are given a choice in retrieval transactions between quick response and most recently updated values.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION

Database management systems and distributed data processing are important approaches currently being developed for control of what we see today as explosions in the amounts of information being handled by computer systems and in the numbers of computer systems available to do useful work. Recently, research on combining these two approaches into distributed database management systems (DDBMSs) has been increasing in an attempt to produce uniform data management policies for networks of communicating computer systems. The goal is to provide users with a general information resource without requiring them to know specific things about how it is built in order to use it effectively (e.g., what data is located on which disk of which computer system). This dissertation develops an evaluation methodology to compare the expected cost/performance of distributed database management schemes in particular operating environments. Four specific examples of schemes (centralized, master/slave, synchronized, and delayed synchronization) are evaluated in detail and compared in terms of average system response time and average network message traffic.

The basic methodology of evaluation begins by analyzing the management scheme and then identifying the specific control paths and the network data flow required to handle both updates and retrievals. The control and data flow information is used to develop a queuing network model of the entire system. Assumptions about the operating characteristics of the system (such as communications connections and delays, processing power, disk rates, and transaction input distribution) are incorporated in the model so that average system response time and average network message traffic can be calculated.

Three of the particular management schemes evaluated are specific cases of standard approaches to database management: a totally centralized scheme, a master/slave scheme wherein one node is in charge of the distributed system, and a synchronized scheme wherein all of the nodes cooperate to manage the database. The fourth scheme, delayed synchronization, is a new proposal that basically consists of delaying the synchronization of update transactions among the multiple data copies of the system. The intention is to improve the average system cost/performance by delaying the resource requirements and communications costs of synchronization until less expensive times. We will see that delayed synchronization is appropriate for applications having only a small probability of updates conflicting and for users who would be willing to deal with out-of-date information if

they could get it more quickly or cheaply.

Applying the evaluation methodology to the particular management schemes illustrates the effects that certain implementation decisions can have on system performance. For example, use of a physical ring to implement the virtual ring for synchronized management imposes sequential communication without the possibility of improvement by broadcasting. However, any cost/performance predictions that are made will need to be verified against data from real systems as those systems are built and have their characteristics fed back into the modeling and analysis techniques used. In the meantime, this dissertation represents a first step toward evaluating the system cost/performance of alternative distributed database management schemes and points out a specific tradeoff in information timeliness that can be used to improve performance in synchronized systems.

The following presentation begins with background and motivation for distributed database systems (Chapter 2) and leads into the new proposals for a timeliness tradeoff and a delayed synchronization management scheme (Chapter 3). The evaluation methodology is developed by treating the four specific cases in detail (Chapter 4). Comparing the results of the performance analyses leads to conclusions about the cost/performance of the schemes in particular operating environments (Chapter 5). Finally, the conclusions are

briefly summarized, several extensions to the work are suggested, and some of the broader implications are introduced (Chapter 6).

Chapter 2

BACKGROUND

The purpose of this chapter is to provide some general background information for the reader who is just being introduced to distributed database management. Some motivation and research goals are presented, along with a brief description of the general problem areas. Additional detail is contained in Appendix A. The more knowledgeable reader is invited to proceed directly to the next chapter.

2.1 DISTRIBUTED DATABASE MANAGEMENT

Distributed databases are often categorized according to how the data are distributed among the nodes of the system. At one extreme there is the partitioned database, which means the data are divided up among the nodes without any duplication at all. At the other extreme is the duplicate copy database, where a complete copy of the entire database is located at each node. Between these two extremes is a huge variety of schemes where copies of some portions or all of the database are located at some or all of the nodes. In order to begin to realize the potential that distributed database systems offer for improved availability and reliability, we

will not consider the partitioned database. We will refer to the rest of the possibilities generically as multiple-copy databases, and only call out the extreme case of complete duplicate copies at each node when the distinction is important.

Once the database is distributed, we can consider whether or not, or how, to distribute the management system which controls it. The two kinds of approaches that have been proposed are:

* the master/slave approach: a single, distinguished, master node is in charge of managing the database and directs whatever management activity is to be done at the other, slave nodes; and
* the synchronized approach: all of the nodes cooperate on an equal level of responsibility.

We can see immediately that both of these approaches will require many messages to be sent, either between master and slaves or among the peers, to accomplish the coordinated management. If the nodes are physically very far apart, the delay due to the communications may become quite noticeable to the user. In addition, we quickly suspect that the master node may be a bottleneck due to update contention for resources and that the overhead involved in coordinating activity among peers may be considerable.

2.2 MOTIVATION

The general goal of the research on distributed database systems is a combination from the database management and distributed processing areas:

* to manage the entire information resource of the distributed system as a whole in a uniform and integrated fashion (i.e., the database management goal), and
* to do it in a way that takes particular advantage of the multiple instances of the various hardware and software resources that occur when individual computer systems are linked together by a communications network (i.e., the distributed processing goal).

This goal is sometimes expressed more specifically from a user viewpoint in terms of physical data independence, availability, and reliability. By physical data independence, we mean that a user of the distributed database system does not need to know anything about specific physical locations (e.g., what disk on which node) or storage layout schemes for the data of interest to him. A DDBMS would be responsible for translating the user's logical request into the ultimate physical storage commands required to access the data, and thus provide the data independence being sought.

By availability we mean that the system has the resources available to do the user's job right now: the necessary data is accessible and there is a processor (or even several

communicating ones) ready to do the work. A DDBMS has the potential to offer improved availability by providing multiple copies of data resources so that contention over sharing single copies can be reduced and by providing multiple processing resources so that a total system workload might be divided into pieces which could be independently or cooperatively processed in parallel.

By reliability we mean that the system as a whole can continue to operate even if some of its individual components should fail, requiring repair and then reincorporation into the system. A DDBMS may provide higher reliability if multiple copies of data resources can be used as on-line back-ups in case one copy fails and if multiple processing sites can be used as on-line back-ups in case a node fails. Such a back-up capability obviously depends on some flexibility in routing communication and user requests around crashed nodes and on whether duplicate data can be made accessible to back-up nodes (such as by remote requests if complete duplicate copies of the database are not kept at each node).

2.3 SOME PROBLEM AREAS

The research that has been done on distributed database systems has identified the following general problem areas (which are described in more detail in Appendix A):

<u>integrity:</u> correctness of individual data items in the context of the whole database as a model representing some enterprise, where a <u>data item</u> is the smallest independently accessible unit of data;

<u>organization:</u> location and arrangement of data and directories to facilitate efficient responses to user requests;

<u>security:</u> protection of data from accidental or malicious corruption and prevention of unauthorized access;

<u>data incompatibility:</u> limitations on data usage because of its structure or representation;

<u>reliability:</u> availability, failure recovery, explicit indications of database status with regard to individual operations, and prevention or reduction of situations where data are inaccessible;

<u>implementation experience:</u> what is required to build systems for actual use; and

<u>cost/performance:</u> how effectively the system responds to the users at what cost.

The state of the art in distributed database systems at the time that the research for this dissertation began can be briefly summarized as follows:

* In general, data incompatibility was being ignored, often by consideration specifically of networks of homogeneous processing nodes.
* All of the other areas except security and

cost/performance had been or were being addressed at various levels of detail.

The research tended to focus on integrity, mostly in terms of concurrent access control; what was required to manage the database so that it operated correctly in the sense that the data would be consistent and that the users would get the results that could be expected by some universal, omniscient observer who could see, in a single frame of time reference, all of the activity that was relevant to the database system. The mechanisms being proposed were complex in detail and it was difficult to establish any common ground among them to serve as some basis for comparisons.

The variety and complexity of the proposed management mechanisms and the lack of published comparisons for cost/performance attracted this author's attention and provided the direction for this research. The question of particular interest was, is there a specific management alternative whose improved cost/performance can be demonstrated in an appropriate environment?

2.4 DEVELOPMENT OF THE PROBLEM

One way to develop a management alternative is to start from a different point of view about what the rules of operation ought to be. Database systems have always been set up to provide users with data values which represent the most

recent information available when the retrieval request was made or received. In frequently updated multiple-copy distributed database systems, however, it will cost both processing overhead and communication delay to handle the complexity of keeping multiple copies synchronized. If we could reduce the processing and communication requirements by not keeping all of the copies completely up-to-date, we could improve the response time and reduce the amount of message traffic. This alternative would be appropriate if the database users would be willing to accept less recent information that could be gotten quickly and/or cheaply, and would be willing to wait for the (probably) more expensive and time-consuming retrieval of the most recent data when it was necessary.

The assumptions on which such an alternative approach would be based are that:

* access to local data is quicker than access to remotely stored data, and
* access to local data is cheaper than access to remotely stored data.

The first assumption is true for distributed systems which are made up of nodes geographically spread out and connected by "thin-wire" communications networks, like the ARPAnet [DAVI73], that have limited bandwidth and provide communications between nodes that are slow relative to communications within a node. The second assumption is true

for the kinds of network pricing policies which are prevalent today, such as for packet-switching nets like Telenet [SOLO78].

To summarize briefly, the specific context and potential payoffs for the research of this dissertation consists of:

* multiple copies of data for improved availability and reliability,
* multiple sites of control for improved availability and reliability, and
* geographic separation of nodes for improved system cost/performance through reduced contention, communications cost, and delay.

Within this context, the particular goals are to develop a management alternative and demonstrate its improved cost/performance.

Chapter 3

AN ALTERNATIVE MANAGEMENT APPROACH

3.1 THE GENERAL SETTING

In order to be specific about the proposals of this dissertation, we will first look at two potential examples to give us the context within which to describe the new alternative approach.

3.1.1 Airline Reservation System

An airline reservation system handles flight scheduling, passenger check-in, and airline management as well as seat reservations. The basic airline reservation system in use today is a large centralized database system. Redundant hardware at the central site ensures minimal possibility of the system being down due to failures. Terminals from all over the country access the central database, usually through a regional concentrator. Terminal, or concentrator, service is provided by polling from the central site. Special protocols have been defined so that various networks can be hooked together for wider applicability. For example, the SITA (Societe Internationale de Telecommunications

Aeronautique) network was established (1949) as a world-wide low-speed message switching network [DAVI73]. When reevaluated in 1964, SITA was reorganized by establishing high-level network centers to join areas of smaller message concentration, so that a network of star networks was formed. New York, London, Paris, Amsterdam, Brussels, Rome, Frankfurt, and Madrid were chosen as high-level centers. The high-level network (completed in 1970) was designed for and has achieved faster, cheaper communications service.

A possible distributed database system for airline reservations would have one file per flight with an identifier consisting of the flight number and date. Copies of the file would be located at regional reservation centers (for the U.S., probably about six of them: north- and south- east, central, and west) instead of just having regional concentrators. One goal would be to minimize response time and communications cost between numerous inquiry terminals and the distributed database (DDB). Some time after successful flight completion, the file would be archived and the copies deleted from on-line storage.

Other types of reservation systems, such as for hotel space, theater tickets or sports programs (with wide appeal such as the World Series), could be set up similarly to the airline system. Minor variations in policy will have to be checked for impact on the DDBMS. Over-booking an airline

flight may be common practice, for example, while selling the same seats twice for a play is not tolerated.

3.1.2 Equipment Supply System

An equipment supply system handles inventory control, order entry, scheduling of deliveries, and aids company management. Existing systems tend to fall into two categories: completely centralized or master/slave copies with deferred nightly updating. For example, a completely centralized system is used by J.I. Case [IBM 77], manufacturer of agricultural and construction vehicles, to manage its spare parts inventory and distribution. The system consists of a central site and 85 terminals in assorted warehouse (one main and ten regional) and office (corporate, distributors, dealers) locations. Order entry, order inquiry, purchasing, receiving, depot replenishment, warehouse control, and management forecasting are all handled on-line.

An example using nightly synchronization and updating of duplicate information is the hierarchical network used by Celanese to handle its inventory and product distribution [WATK77]. Two large central computers divide up the responsibility of being master by partitioning control of the data: one handles inventory control and bills of lading, and the other handles invoicing and accounts receivable. These two are both connected to a smaller computer which controls an

automated warehouse and manages a set of small production control computers. Updates which have to be distributed to, or synchronized among, various locations in the network are batched for nightly shipment both up and down the hierarchy; e.g., after all data are updated and synchronized, one of the central computers will prepare a list of everything to be shipped on a particular day, and then send it to the warehouse control computer for coordination, scheduling and handling.

The more general distributed system being suggested here would be appropriate to a large company, for which the country would be divided into regions, each with a regional headquarters and sales force. Warehouses would be responsible for their own local inventory databases, with copies located at the appropriate regional headquarters. The regional headquarters would also maintain sales, customer, and delivery information for the region, and the sales force would require copies of inventory located in adjacent regions.

There are numerous variations possible on a system of this basic type. Some other applications would include inventory control, accounting, and facilities management for: supermarket chains, large multiple-plant manufacturing operations, department stores, multi-branch libraries, and utilities such as electric or telephone companies.

3.1.3 Cost/Performance Concerns

In considering the examples from a cost/performance viewpoint, we immediately notice two things:

* numerous messages will be required to keep all of the data copies consistent (so that users accessing different copies concurrently do not get different results, for example), and
* the delays involved in both the message transmission and the processing required to coordinate the activity among the nodes may be significant to the user.

"Thin-wire" communications among nodes, i.e., limited bandwidth that is slow relative to communications within a node, is of particular interest because it is a common characteristic of many of the networks existing today. Even as special leased telephone lines and satellite systems begin to offer alternatives to old speed and bandwidth restrictions, their cost will continue to make them impractical for many companies and applications, at least over the next five years [MAND78].

The major database communications bottleneck within a node is usually between memory and disk storage. It is described by two parameters: set-up or access time and sending or transfer rate. The large disks suitable for database storage typically provide average access times on the order of 20 msec and transfer rates on the order of 1 microsec

per byte. In situations where disk utilization gets high enough, contention may be manifested in long waits just to initiate access.

Communications among network nodes can also be characterized by access time and transfer rate, but now transit times also become appreciable. Current long-range communication (5 - 50 Kilobaud) provides transfer rates of roughly .2 - 2 msec per byte, and transit times (e.g., across the U.S.) on the order of 10 msec per byte. Remote access time is defined as the interval between initiation of the call for the remote data and transfer of the first byte. Typical times are 10 to 100 times slower than local disk accessing (for example, ARPAnet times to establish a virtual circuit). This gives a rough estimate for remote access time that is on the order of a second. Computer Corporation of America is in the process of designing a distributed database system that is based on a similar assumption that access to the first byte of data at a remote node will be 50 to 100 times slower than access to data on a local disk [WONG77].

We can summarize our concern over communications and coordination costs and delays with two quantities: the number of network messages to complete a user transaction represents the cost aspect and the response time which the user sees (between completing his request and receiving his reply) represents the performance aspect.

3.2 A TRADEOFF IN TIMELINESS

3.2.1 Definitions

Let us assume that for certain applications distributed data copies would not have to be immediately synchronized, and propose that therefore the users of the database would get quicker response time and the communications costs in the system could be lower. The improvement in response time would be based on the premise that users would be willing to work with old data (out of date with respect to the database as a whole) if they could get it quickly. Correspondingly, the lower communications cost would be based on the premise that periodic batched transmission of updates would be cheaper than handling each one as it occurs. Both of these premises are supported by experience with existing centralized database systems such as the Retail Store System by IBM [MCEN75], where updates are collected throughout the day at individual stores and then batched to company headquarters at night. Similarly, Celanese [WATK77] does their synchronization daily rather that sending current updates through the network during the day. If absolutely up-to-date information is required, it must be handled by making a telephone call to a person in the appropriate warehouse.

We will explicitly define "data timeliness" as how recent (or how old) the information is. By associating a timestamp with every data item as it is entered into the database (insertions and updates), we will be able to quantify the data timeliness as the difference between current time and the item timestamp. In this way, data timeliness is an intrinsic attribute of a data item which depends on the context from which it is referenced. We will also define "access timeliness" to indicate how quickly the information is needed in order to be useful to the one who requested it. The user will subjectively define some time interval within which he needs the information in order for the access to be timely.

3.2.2 QR-BI Retrievals

The tradeoff between data and access timeliness is made available to the user as two types of retrievals:

* quick response (QR) retrievals for the user who values access timeliness more, and
* best information (BI) retrievals for the user who requires the best data timeliness.

Interactive users, for example, who want quick response for retrievals, and are willing to deal with data values that may not reflect the most recent update cycles, are served quickly. In contrast, batched programs which can easily (or transparently, at least) afford to wait for the most recent data values to be found get the best information possible. QR

users will be given the additional option (if included at database design time) of determining the timeliness of their retrieved data by looking at the associated timestamps.

The operating difference required by the users accessing the distributed database is minimal; instead of having simply a RETRIEVE operation, there would be (at least) two options: RETRIEVE QUICK (or RETRIEVE QR) and RETRIEVE BEST (or RETRIEVE BI). An intermediate option of RETRIEVE BEST WITHIN (time limit) would also be possible, but could be more complicated to handle: first a definition of what "best" means within a time limit would be needed, and then a strategy for how to determine or calculate it would have to be set up. Standard defaults could be arranged, so that, for example, interactive users would get QR unless they specified BI, and batch uses would get BI unless QR was explicitly requested.

To deal with the retrieval options most productively, a user must understand something about the possible responses to his requests. A QR request may return the best information if the local copy happens to be the last one updated. A BI request may not return the absolute best information, in the context of the distributed database as a whole, if some node is down or too busy to reply; the best information available when the request is processed (that is, the relative best) will be what is returned. On the other hand, the intermediate option of best within a time limit may produce no result at

all if the data are not local and the limit is reached too quickly for a remote result to have arrived. The user could then be given a choice of removing the limit or restating his request (e.g., continue waiting for the result to arrive or cancel the request by throwing away the result when it does arrive). For simplicity, this dissertation will deal only with the extreme options, QR and BI.

It is important to remember that this tradeoff between QR and BI is a retrieval option and does not affect the updating policy except from a performance standpoint. If it is important to the users of a particular application to know with complete certainty that their updates have been properly applied, the database management system will have to do whatever locking, coordination, and verification is required. The only problem is that in geographically dispersed systems, such activity may involve many resources and require appreciable amounts of time.

Some simple examples of the differences between QR and BI requests may help to motivate the value of the tradeoff involved. In an airline reservation system a manager might want to know approximately how many seats were already booked on a particular flight for next month. A QR retrieval would be appropriate for the "approximately" and the manager would receive his answer quickly from local data. If Joe Green wanted to know whether his secretary had properly booked him

on that flight, a BI retrieval would get him the answer with certainty. Similarly, a regional salesman asking for general information purposes how many units of a particular type were available in the warehouse closest to him would use QR and generate as little database traffic and contention as possible. The circumstances surrounding an individual request can have a very definite effect on which type of retrieval is appropriate. For example, a question about whether a particular order had been filled and actually shipped could be handled by QR and a timestamp if the salesman is checking for his own information, but had better be retrieved as BI when the customer is waiting on the telephone to find out.

3.2.3 Data Quality

Since we expect retrieval situations where the number of QRs far outweigh the number of BIs to give us a significant cost/performance improvement for the distributed database system as a whole, we would like to provide even more incentive for the use of QR retrievals. The timestamps associated with each data item can sometimes be used to establish the appropriateness of a particular data value retrieved with QR by defining its timeliness. Suppose we take a broader view and consider something we will call data quality. This would be an indication of what changes had taken place since the local copy of the data item had last been updated. Data quality would be most appropriate, then,

when associated with a fairly large unit of data, so that the overhead of its storage would remain relatively small. This is in contrast with the timeliness approach of attaching a timestamp to the smallest unit of data which can be updated. The issue of granularity, i.e., the question of what size unit, is similar in both cases. The maximum information is gained with the smallest unit association, but at maximum cost in storage. The timeliness approach assumes a willingness to pay the storage cost for the timestamps; the granularity for the quality indicator can be considered separately, and depends specifically on the type of data and type of application involved.

In the distributed database context, when a user closes or releases a file (i.e., the subset of the database being worked with) that he has updated, he could be asked by the DDBMS for an evaluation of the importance of the changes to the file during that session. The DDBMS will add a timestamp to that evaluation to create the "quality" information to be associated with that update session on that file. Such a quality indicator could be sent by the DDBMS to any copies of the file which are not being synchronously or immediately updated.

The value of data quality indicators to users of the data other than the updating one, depends on a number of considerations such as: the trustworthiness of the one

assigning the evaluation, the meaningfulness of the timestamp outside its frame of reference (if node clocks are not synchronized in some way, they represent completely independent frames of time reference), the degree of agreement among the user community on what's important, and even the relationship between what constitutes a revision and the number of update sessions required to produce that revision. These problems emphasize the difficulties inherent in the paradoxical attempt to quantify quality.

3.2.3.1 Airline Reservation Example.

A simple way to use data quality indicators in the airline reservation system would be to associate with each flight file a count of how many seats remain unreserved throughout the network. Then, even if the local file copy did not have all of the information regarding each reservation that had been made, new reservations could be made with less chance of overbooking the flight. The detail of such data quality use requires more information about how the system operates (see section 3.3.2.2).

3.2.3.2 Equipment Supply Example.

A similar use of data quality would be appropriate in an equipment supply system. If local copies of inventory files for warehouses not in the local region were not kept

immediately up to date, the data quality indicator could be used just to signal that update activity had occurred on a particular item. For more detail, see section 3.3.2.3.

3.2.3.3 Text Example.

A very detailed example of the use of data quality which can be developed with no additional operational rules is for text editing, where updating activity usually produces distinct versions of the text file. I might give you yesterday's version of the text to read, telling you that even though it is not a copy of the most recent version, the changes that were made were all minor editing changes which had little or no effect on the text content and meaning. You would probably be willing to read and comment on that old version, with a certain amount of confidence that your copy is a reasonable representation of the actual, current text.

A data quality indicator (DQI) for text files can be defined as a timestamp (TS) and a severity-of-change (SOC) code: DQI = (TS, SOC), where the SOC code indicates how many changes were made and how important they were considered by the user who made them. One way to implement the code as three digits is shown in Table 3-1. Let us consider, for example, an author editing a textbook where the data unit is the whole book. For rewriting an entire chapter, he might assign a SOC code of 133 to mean one large chunk was changed

significantly; for a section, 122 or 123 to mean one medium chunk changed a medium or major amount; and the timestamps would indicate when the last update of the revision was made. For changing all words "which" to "that", the SOC code probably would be 311. The particular advantages to this scheme are its simplicity and its brevity (only four or six bits are needed to hold the code).

It is not likely that this type of SOC code can be generated automatically by the DDBMS during an update session. Changes could be counted, but chunk size and importance would be too difficult to keep track of and to evaluate. Despite such philosophical considerations, we can see that a data quality indicator would be of particular use for any files that are not updated immediately in a distributed database. The DQI would be transmitted at the end of each update session to the remote copy sites and added to the file header as part of a change history until the copy actually receives the update set summarized by the DQI. There will be a definite tradeoff to be examined between the length of the history in the file header and the delay period of the updating. In order to be certain when the DQI could be erased, its timestamp should be that of the concluding update transaction. When the copy site later receives a set of updates with timestamps up to and including that of the DQI, the DDBMS will know which DQI is no longer needed.

Table 3-1. Severity-of-change (SOC) code

SOC = (CNO, CSZ, IMP), where

CNO: number of data chunks changed,

CSZ: size of data chunks changed,

IMP: importance of changes.

	digits		
values	CNO	CSZ	IMP
1	one	small	minor
2	a few	medium	medium
3	many	large	major

* a value of 0 for any digit means it was not specified

3.3 DELAYED SYNCHRONIZATION MANAGEMENT

3.3.1 A New Management Approach

In addition to the timeliness tradeoff, we will need the following assumptions in order to proceed with the proposal of an alternative management scheme that delays synchronization in a frequently updated, multiple-copy distributed database:

* the cost for a single batch of N messages is less than the total cost for N individual messages, and

* data copies can be allowed to diverge (i.e., become inconsistent) because BI retrievals can always find the most recent value for any data item whenever all the network nodes are up.

Delayed synchronization management consists of handling updates immediately at the site of origin and not synchronizing them with any other data copies in the distributed system until after some delay period. The length of the delay will determine how much the copies diverge according to how many updates come in and to what sites. The specific objective is to improve the average system cost/performance by delaying some of the expensive and time-consuming resource requirements (to synchronize the updates among all data copies) until less busy, less expensive times.

3.3.2 Fundamental Notions

The first step in a timeliness approach to delaying the synchronization of multiple copies in a distributed database is to classify the multiple-copy files according to their requirements for update application and synchronization. This could be a static classification, by the users and the database administrator at design time, or some conditions could be specified for dynamic classification. The classes of files are those for which:

(1) updates are applicable at all copies as soon as possible and must be synchronized , called "synch" files;
(2) updates are applicable to all copies but need not be immediately synchronized, called "delsync" files; and
(3) updates are applicable to only one primary copy and are subsequently indicated (e.g., broadcast) to all back-up copies, called "master/slave" files.

Files of the latter two types, delsync and master/slave, will allow several update policy options such as immediate, delayed, or periodic transmission of updates, with possible intermediate transmission of some sort of data quality indication. All updates will eventually be done to all copies in order to provide the long-term mutual consistency, with the delay period depending on the particular application. Notice that the word "file" here (and throughout this dissertation) refers to some identifiable, replicated subset of the whole database (such as a single relation or a Codasyl area). It does not necessarily imply some particular storage or header structure, or even a data model scheme (hierarchical, relational, network).

The second step of a timeliness approach is to classify the DDB uses according to whether data timeliness or access timeliness is more important. That is, divide the uses or users into groups requiring or desiring quickest response as opposed to best information. This phase could be handled in several different ways, including:

(1) preclassify all known transaction types at design time (this is done in SDD-1, see section A.7.2 of appendices);

(2) preclassify users at system entry time as to whether they want quick response or will wait for latest information;

(3) keep user profiles and classify users according to their use history;

(4) allow dynamic specification of these extreme options and some intermediate options as well: respond within some time limit, or return the best information which can be found within some time limit.

3.3.2.1 Ordinary Operation.

Delsync files will be appropriate only when the delay in synchronization will not jeopardize the long-term consistency of the database. Whenever an update for a delsync file is received, the DDBMS must check to see whether application of the update would violate any consistency constraints (such as not overbooking an airline flight). If constraints would be violated, then synchronization must be initiated and the update held off until all copies are brought up to date and the file is changed to synchronized mode.

If no constraint would be violated, we can define two types of update policies: immediate and delayed. In the

immediate case, the update would be applied locally and sent to other copies, where we again have a choice of applying the update either immediately or after a delay. Delay local to the update originator tends to minimize the average communications cost and delay while risking an occasional penalty if some user requests a BI retrieval. Delay in applying at the copies once the update has been received may be used to keep disk contention minimal during periods of high activity.

For updating delayed at the originator, the update would be saved locally after local application and sent out to the other copies after a delay period. A technique called differential files, commonly used for file storage, back-up and recovery [VERH78], will conveniently handle both the update saving and the access to the file so that updated information is properly retrieved. If updating were not too frequent or if the delay period were long, an indicator such as data quality could be sent to the other copies to show that updates had occurred. The appropriateness of sending such an indication depends on the frequency and size of the updates, and on the granularity of the data aggregate to which the DQI applies. For small, frequent updates, the indicator traffic would be about the same as sending the updates themselves and would completely destroy the advantage of saving the updates for a big block transmission. Intermediate solutions could be created by batching the update transmissions more frequently,

such as during a particular, agreed-upon five minute period out of each hour of operation.

Retrievals from delsync files would be answered locally unless the information was not held locally or a specific request was received for best information. For BI, the retrieval would have to be sent to all file copies, the results collected together, and the latest information selected from among the responses. The user would have explicitly given up quick access in favor of best data.

In general, the approach is to preclassify the types of data according to the probability of conflicting updates and the penalty for old data or possible errors. For files which seem to qualify as delsync, constraints are specified to minimize the chances of errors from conflicting updates, and error detection is coupled with correction or notification procedures. If the file mode is to change dynamically, change conditions and procedures must be established, as suggested above.

In spite of additional complexity, the distributed file copies give good data availability and quicker response to many retrievals. The chief advantage over a centralized system is cheaper, quicker average retrieval since total network traffic and queuing waits are reduced. For interactive management queries, the quick look at local copies of information and the associated quality indication is

adequate for nearly all requests and again saves network traffic, lowering associated communications costs.

3.3.2.2 Airline Reservation System Example.

A classification at design time of the files for an airline reservation system may be: near-full flights as synch type, near-empty flights as delsync type, and checking-in flights as master/slave type. It is easy to see that reserving seats on nearly full flights needs to be coordinated among file copies so that the flight is not over-booked. Reserving seats on near-empty flights can be done in parallel on the multiple file copies with periodic (probably nightly) merging of all updates into all copies. This works well until the number of reservations exceeds some flight threshold. For example, suppose the threshold were set at 50% full. Any reservation which would cause the local file to exceed the threshold would require all copies to synchronize before the reservation could be granted. The file would then become a synch type and all further reservations would be synchronized among all file copies. The particular advantage to this approach is to drastically reduce the average amount of network traffic (by updating delsync files locally and only merging during periods of low activity) so that quick response from the fewer synch files can be provided and communications cost can be reduced. Thus, updates on nearly-full flights close to departure time can still be handled more quickly than

in centralized or fully synchronized types of systems.

We can, of course, construct pathological cases of thresholds, block reservations, and concurrent updates of distinct copies which will overbook the flight:

100 seats on the flight,

total of 12 prior reservations,

threshold of 50,

3 file copies,

3 simultaneous requests for blocks of 30 seats each, yielding allocation of 102 seats without even generating a synchronization requirement at any file site. Thresholds will have to be chosen through experience so as to minimize the probability of such incorrect file operations by taking into account both the seating capacity and the overbooking probability or penalty. A crude threshold value, for example, could be set by dividing the total number of seats by the number of reservation sites; for most files this is too conservative an approach. Another possibility is to add a second threshold on the size of block requests, which could be dependent on both the number of seats still available and the number of reservation sites in the network.

In addition, a method for recovering from any error must also be provided. In the example here, flight over-booking by two seats, the airline may choose to leave the conflict and count on subsequent cancellations or no-shows to resolve it.

In any case, the DDBMS will have to provide some type of notification (such as a terminal alert to an operator) to establish awareness of the error situation. Typically the error will be discovered during the nightly merge operation if the file is still in delsync mode, or during the transition from delsync to synch mode. In either case, the error will appear within 24 hours. To ensure that it also shows up at least several days before the flight, transition can be forced (regardless of any threshold) some minimum number of days before the flight. Such forced transition has the additional benefit that no update will have to wait while the transition takes place.

A file will be changed to master/slave type when passengers begin checking in for the flight. The file copy closest to the check-in point becomes the master and all further updates must go through the master for transmission to the slave copies. This will ensure uniqueness of seat assignments, accuracy of passenger lists, and proper passing of through-routing confirmations for connecting flights.

Classification of uses in the distributed reservation database shows that reservations for near-empty flights get the quickest possible response from local file copies, while reservations for near-full flights are properly synchronized without resource competition from the less urgent requests. Management statistics and planning, usually interactive

terminal operations, will desire quick response -- (as long as archived data is not requested) retrievals can be all from local copies, whether synch, delsync or slave. Report generation requires accurate information, but as a batched operation usually run in the background, it can afford to wait for any information to be gathered either from delsync copies or from archived files.

The object of all this classification is to allow some flexibility in DDBMS strategies for file update and retrieval in order to improve average response time as much as possible and cut communications costs. Near-empty flight files will be handled by periodic update transmissions among the copies, while near-full flight files will be synchronized by an appropriate protocol. Quick response management queries will be answered strictly from local data without waiting for communications from other nodes, while report generation will wait to gather the latest information from all locations containing it.

3.3.2.3 Equipment Supply System Example.

File classifications for equipment supply are of two types, static and dynamic. Warehoused inventory will be preclassified and remain master/slave, as will production (manufacturing) records. Installed inventory records (e.g., for rented equipment) and customer information (e.g., address)

will be delsync, as the low rate of update activity has low probability of conflicts occurring. Billing and accounting files will require synchronization for accuracy and timeliness.

Dynamic file classification applies to equipment orders and service requests. Orders being entered and checked for consistency may be delsync and change to synch as they are assigned production line positions and warehoused equipment. After assigned equipment is produced, as it is assembled for shipping, the filled order may be master/slave for localized control and ensurance of proper delivery. Similarly, service requests may be entered as delsync, assignment must be synchronized (transition occurs nightly so work schedules can be prepared), and completed requests can be delsync as history records (where updates are unlikely, if even permitted).

Salesmen entering order transactions in the system will first request items from their local region inventory. If the items are not available locally, adjacent regions (close physical proximity) will be checked. The warehouse master inventory files will update local regional headquarters slaves, but may only transmit change indications to adjacent regions until off-peak hours when update transmission would be cheaper. The adjacent region salesman, then, would have to give up access timeliness and wait for remote information if he needed the most recent information on an item file that had

a change indicated (DQI), or the indication could include information about how much or what type change had occurred. Delivery scheduling and billing would be periodic batch runs and would wait to be sure their information was completely up to date. Management terminal transactions would be serviced quickly from local copies of information unless the user specifically requested most recent data.

3.3.3 Control Flow

Management by delayed synchronization can be summarized as in Figure 3-1. The language used is not any particular one, it is just a combination of structured primitives and brief explanations.

Figures 3-2 through 3-5 represent the next level of refinement of the overview control flow and they begin to incorporate some of the details of the management mechanisms for delayed synchronization.

The object of Figure 3-2 is to make clear that the delsync updates are handled locally, and account for the divergence of the file copies from the last time they were synchronized by the merging process. One of the premises of delsync would have to be a high degree of locality of reference; that is, that nearly all of the retrieval requests at a particular node can be satisfied by data at that node. The few requests that will need to be forwarded to other nodes

Figure 3-1. Overview of Delsync Management

```
DO CASE (transaction type):
      CASE (update)
             IF (no constraint violated)
                    THEN delsync update
                    ELSE make transition
                         synchronize the updating
             ENDIF
      ENDCASE
      CASE (retrieval)
             IF (BI requested)
                    THEN get data from all copies
                         select best result
                    ELSE get data (& DQI) from
                            closest copy
             ENDIF
             process result for presentation to user
      ENDCASE
ENDDO
```

are handled as in Figure 3-3.

The details for processing BI retrieval requests are shown in Figures 3-4 and 3-5. The broadcast method is probably the quickest way to collect data from all the other nodes, but the use of timers shows that there may be difficulty if any nodes are so busy that they cannot respond in time or if a node has actually crashed. The broadcaster cannot tell the difference in general, although it is possible to require busy nodes to send some appropriate status indication that will usually be guaranteed to arrive within

Figure 3-2. Delsync Update

```
copy update into deferred merge file for later transmission
"apply" update by placing it into differential file
        associated with main file copy (all access
        to file is directed first through the
        differential file)
```

the time limit. The fact some node did not contribute to the BI collection process should probably be communicated to the user who made the BI request. The result returned to him will only represent the best information currently available in the system, not necessarily the actual most recent data value assigned.

Figure 3-3. Get Data From Quickest Copy

```
DO CASE (data location):
        CASE (local)
                get data
        ENDCASE
        CASE (not local)
                broadcast request for data
                result <-- first return
        ENDCASE
ENDDO
```

Figure 3-4. Get Data from All Copies

```
set timer
broadcast request
DO UNTIL (timer runout or all responses are in)
        save result returned
        check off responding site
ENDDO
IF (timer runout)
        THEN    alert for possible crash
                note non-responding sites
                        for later recovery
ENDIF
```

Figure 3-5. Select Result

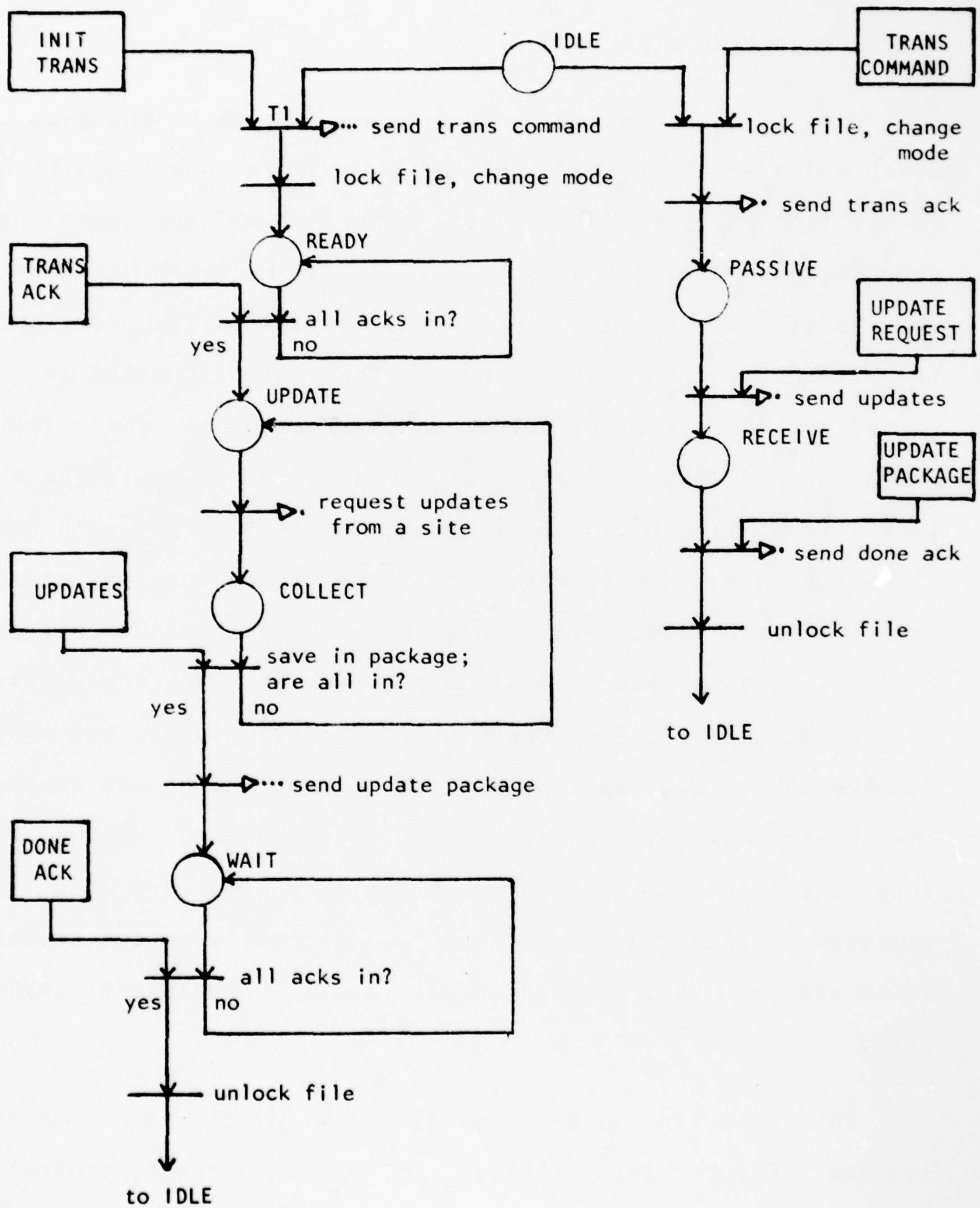
```
find response with latest timestamp *
result <-- response with latest timestamp


* consider two timestamps, TS1 and TS2:
        TS1 = ( site-id1, clock1 )
        TS2 = ( site-id2, clock2 )

TS2 is "later than" TS1:
        IF clock2 > clock1
        OR      IF clock2 = clock1
                AND site-id2 > site-id1.
```

3.3.4 Transition Algorithm

When it is necessary for a delsync file to be changed to synch mode, an algorithm is needed to handle the transition. The polling solution is a simple master/slave type approach: the node discovering that a transition is required (call it the initiator) takes all of the responsibility for that transition. The initiator sends a transition command to all copies so that all file copies can be locked and have their modes changed. Upon receipt of transition acknowledgements (acks) from all copies, the initiator collects from each copy all of the updates that have been saved since the last merge. The initiator does all consistency checking and conflict resolution, assembles a total update package with the updates in correct serial order, transmits it to all copies, and waits for the acknowledgements of receipt. When the copies receive the update package, they apply the updates and unlock the file; the initiator must wait until all copies are done. It can then unlock its own copy and proceed with the update, which will now be processed on the (newly established) synch file.

This solution may be symbolized with an evaluation net diagram, Figure 3-6 [ELLI77]. In this format, circles represent possible states or "locations" of the procedure being symbolized, rectangles represent incoming "message locations" or queues, and horizontal lines stand for

Figure 3-6. Polling Solution for Transition



transitions. Transitions are "fired" by having tokens on all input locations of a transition. Transitions may involve the sending of new messages, represented by a hollow arrowhead on a transition line. The dots following the arrowhead indicate whether a single (one dot) or multiple (three dots imply two or more) receivers are intended. Transition actions are explained as comments alongside the lines.

In Figure 3-6, then, the initial state consists of an internal token on IDLE. When a file mode change from delsync to synch is required, a token arrives on INIT TRANS and transition T1 fires. The transition command is sent to all other nodes, the file is locked and has its mode changed from delsync to synch. When all nodes have signified ready with their acks (a token is placed on TRANS ACK each time the communications subsystem receives an acknowledgement from another node), the initiator polls the sites to collect all updates saved since the last merge of the file copies. When the update package has been received by all nodes, the file may be unlocked and any pending updates may proceed on the newly synchronized file.

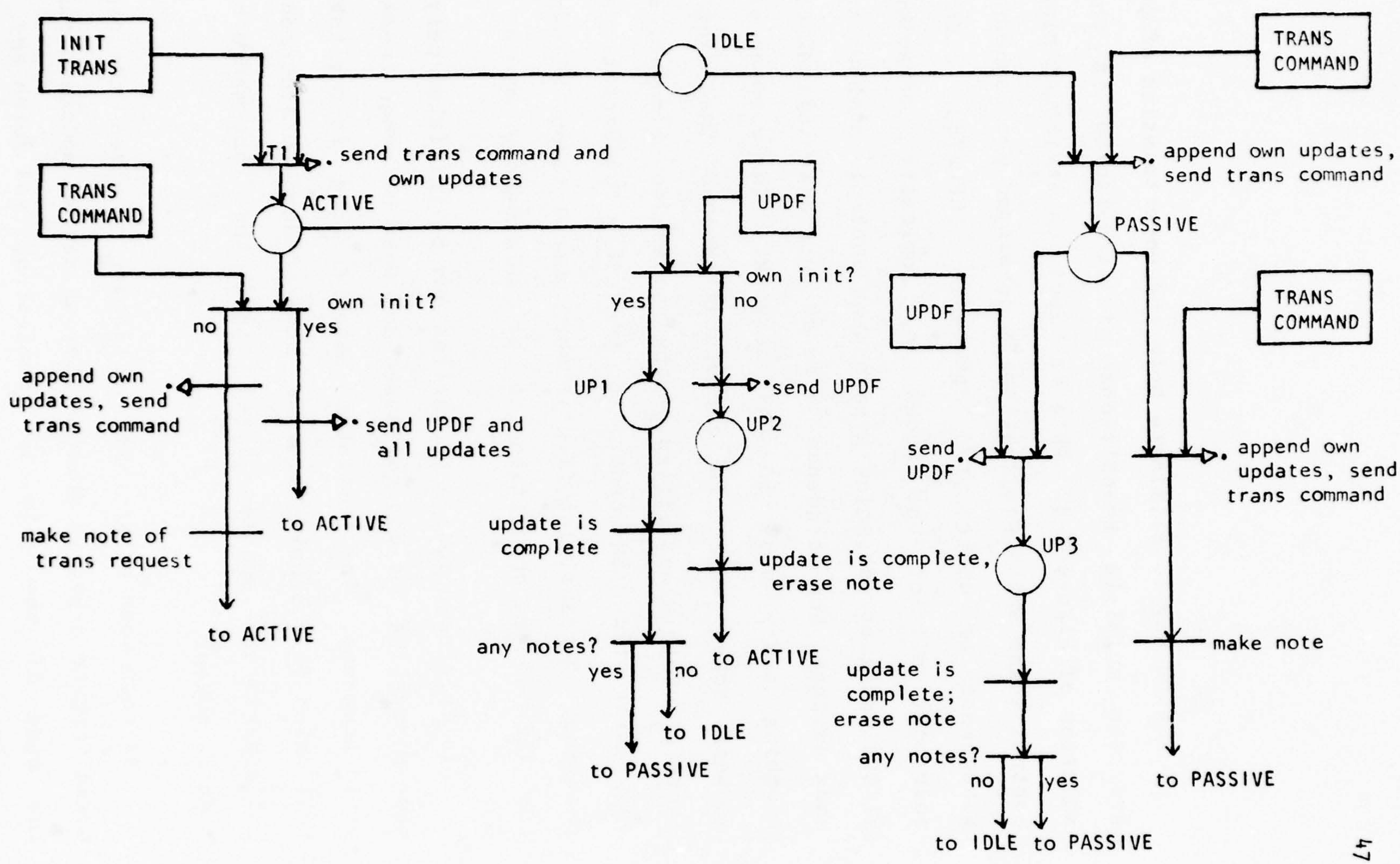
There are a number of difficulties with the polling solution for transition as it is described. A big disadvantage is that the algorithm takes an unpredictable amount of time while waiting for acks and updates. It also cannot distinguish between a node that is slow in answering

and a node that is down. Even if the "down" node has truly crashed and is not at all operational, it still probably has unshared updates saved since the last merge. If crossing a threshold constraint initiated the transition, it could be dangerous to synchronize and proceed without the crashed node's outstanding updates. Thresholds would have to be established so as to minimize the probability of an inconsistent result.

Another candidate for a transition algorithm, based on a ring solution, is shown in Figure 3-7. In this case, the transition initiator creates a transition command and appends to it the updates (from that node) which have been saved since the last merge. The combination is passed to the next node, which appends the updates it has saved, and then on around the logical ring one node at a time. By the time the command returns to the initiator, all the updates required for the merge have been collected and the entire package can be circulated for application. Notes are used to keep track of multiple transitions in progress simultaneously.

A big drawback to the circulation scheme is that there is a possibility for multiple nodes to initiate transitions on the same file at the same time. For frequently updated files nearing the transition threshold, this undesirable property cannot be dismissed as improbable.

Figure 3-7. Ring Solution for Transition



A good way to eliminate both the uncertainties of polling and the multiple transitions of circulation is the ring solution of Figure 3-8. This is based on a virtual ring with control token which was originally developed for synchronized management of distributed databases [LELA78]. In this solution, all of the nodes are logically connected in a virtual ring by assigning a permanent control number to each node so that the sequence creates a single logical circuit touching every node in the network. The purpose of the control token which is circulated around the ring is to restrict certain activities of the nodes to a one-at-a-time sequential flow in order to prevent concurrent access problems. A transition flag in the control token allows only one file mode transition to be in progress at a time.

In this solution, a transition may be initiated only by a node possessing the control token and when no other transition is in progress. The initiator sets a flag in the token to tell which file should have its mode changed to synchronized, and appends the updates saved since the last merge to the control token.

As each node after the initiator receives the control token (Figure 3-9), it appends its own saved updates, changes the mode of the file in transition, and notes any down successors in the ring by adding their names to the list in

Figure 3-8. Virtual Ring with Token Solution for Transition

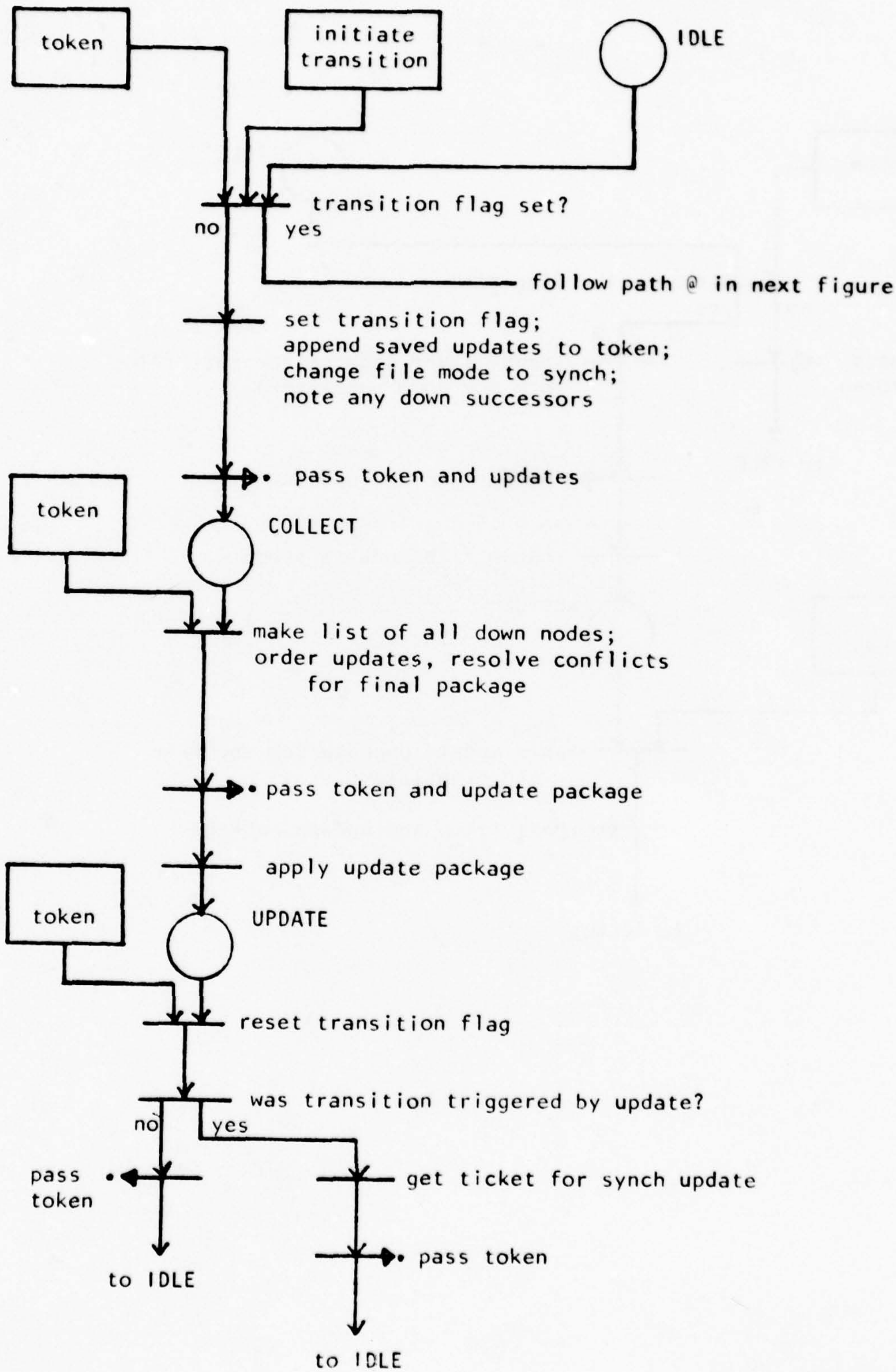
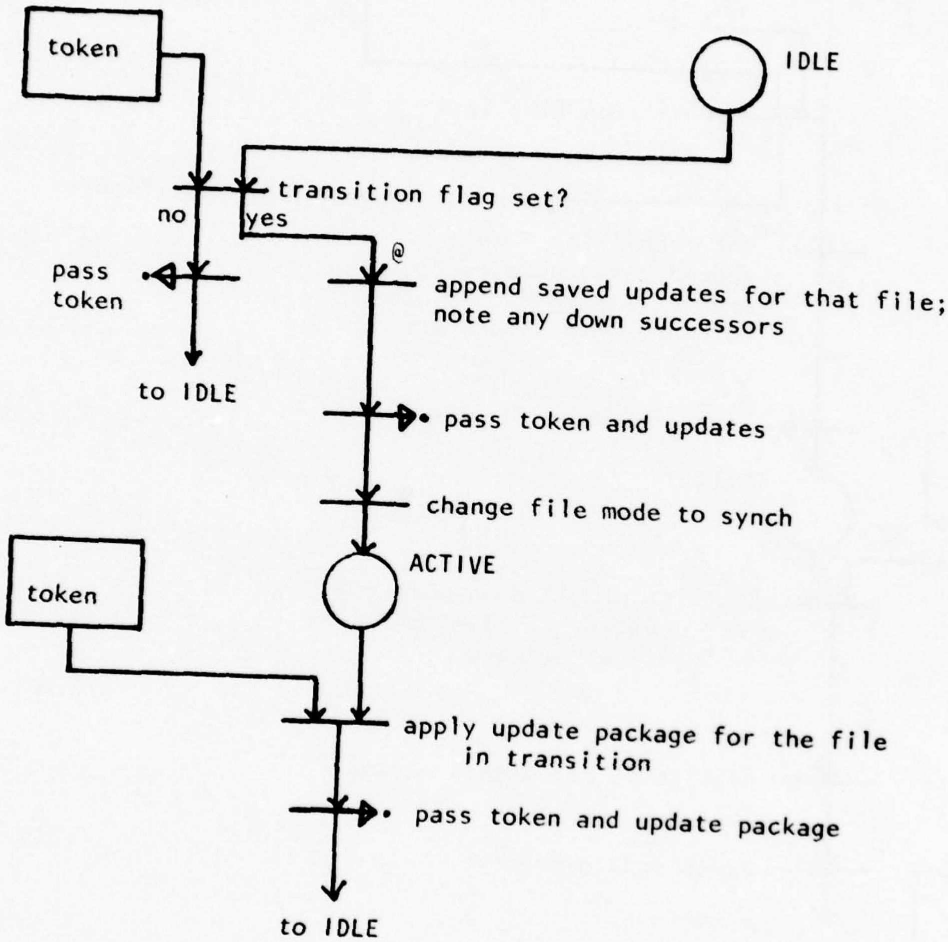


Figure 3-9. Virtual Ring: Non-initiating Node View



the control token. The control token and updates are passed to the successor. Subsequent arrival of the control token at a non-initiating node implies all available saved updates are present in the package and can be applied in timestamp order. The file is then unlocked to local updates and the control token is passed to the ring successor.

The virtual ring incorporates timers to maintain circulation of the control token itself. Control is relinquished only on positive acknowledgement of receipt of the control token. If a timeout occurs before the ack arrives, failure of the successor is suspected. This mechanism would be used to create the list within the token of down nodes or of any nodes failing to participate in the transition of a particular file. The list would be kept by the initiator, along with the total update collection, for later use in bringing non-participating nodes up to date. Similar timers could also be put into the polling solution.

A disadvantage to the virtual ring solution is that a node must wait for the control token to arrive before it can attempt to initiate a transition. Furthermore, if the token has a transition in progress, the attempt will have to wait at least one more control token circuit of the ring. In the worst case, the wait could involve N-2 (N nodes in the ring) circuits if the successor initiated the current transition and each other node also had a transition waiting.

We will choose the virtual ring solution for transition, despite its variability of execution time, because of its explicit control over the transition process and the status-checking mechanism inherent in the circulation timers.

One particularly interesting problem occurs in any of the transition solutions if a "down" node has not really crashed but is just disconnected from the network. That node could continue (by design or by accident) to accept updates to its local copy of the file, which is still in delsync mode. This is a special case of network partitioning, which we will call single-node disconnect. If it is deemed serious enough, data quality indicators can be exchanged among copies between merges so that the probability of error due to exceeding thresholds is minimized. For example, in the airline reservation system, each indicator could contain the number of seats reserved on each update. The copies could keep a running tally of the number of seats reserved. Then even if a node is down at transition time, the number of seats held by updates outstanding before crash would be known.

Single-node disconnect can at least be recognized in the virtual ring with token if another timer is reset to some maximum allowable delay (e.g., the time required for a complete circuit of the virtual ring) on each token visit. A time-out before token arrival should trigger the suspicion that the node with the time-out is disconnected, and a

decision could be made (pre-programmed or interactively with an operator) on whether to discontinue delsync updates as well as synchronized and master/slave. Notice that this timer is in addition to the one required in waiting for the acknowledgement that a passed control token has been received.

3.3.5 Merging

The algorithm for merging is similar to that for transition except that the file mode is not changed. The initiator sets a flag in the token to tell which file is being merged, and appends the updates saved since the last merge to the back of the token. Each successive node around the ring appends its own saved updates to the collection circulating with the token. The collection is complete when the token gets back to the initiator, and the second circulation serves to distribute all of the updates to all of the nodes. Upon returning to the initiator again, the merge is complete; the collection can be removed and the token flag reset for some other node to use. If some node is down when the merge occurs, the initiator will be responsible for saving the entire update collection for transmission at recovery time.

3.3.6 Update Conflicts, Error Notification and Recovery

Delsync files have been predicated on a small probability of conflicting updates. Let us consider what to do in case

conflicts do occur.

The first type of conflict is multiple updates to the same item. Differential files are appropriate for handling this aspect of delsync files. Local updates are kept in the difference file between merges. If at merge time the updates from all other nodes are also inserted into the difference file, conflicts can be discovered and ordered for application according to their origination timestamps. To complete the merge operation, the entire difference file is applied to the main archival file at the appropriate time (see the previous section and Figure 3-9).

The second type of conflict is violation of delsync constraints; it will have to be checked for during each merge. This is the case where none of the individual files has triggered the transition from a delsync- to a synchronized-type file, and yet in aggregate a constraint is violated (see the airline reservation example in section 3.3.2.2). The first thing the DDBMS must do after discovering such a conflict is to provide an error notification. The questions are, who should be told, how, and what does he need to know. The answers depend on what recovery procedures are appropriate to the applications using the file. If the error is to be tolerated (as in overbooked flights), error notification may consist of just a message printed in an operator log at a specially designated node. The "recovery"

action would then be to mark the file in violation and allow thereafter only those updates which would counteract the violation (e.g., cancelled reservations) until the error was cancelled out.

A real recovery might be possible if the constraint violation were detected during collection of the updates for merge (i.e., before application of the difference file). The updates could be listed in reverse order by timestamp and the last few examined to see if rejecting them (albeit, belatedly) would prevent the conflict. Again, the application would determine the appropriateness of what is, in effect, backing out the last transactions. It would certainly not please most clients to have a transaction accepted one day and rejected the next. The cost of such a possibility would have to be considered very carefully when first choosing a delsync management approach and then setting the delsync constraints and transition triggers.

Chapter 4

EVALUATION OF THE PROPOSED APPROACH

4.1 INTRODUCTION

It is common to evaluate new management schemes by doing some experiments to make comparisons with schemes which are well understood and documented. In the case of distributed database management, that is an impossible task because there are no such things. The implementation of general-purpose distributed database management systems is just beginning with systems such as INGRES and SDD-1 (for details see section A.7 of the appendices). They are not fully implemented and have not been designed with test bed functions in mind so that new algorithms or schemes can be plugged into the system in place of the original ones. It is certainly outside the scope of the research for this dissertation to build an appropriate test bed, or even to try to coordinate with the developing implementations in order to provide the basis for direct experimentation.

Fortunately, we do have an alternative approach that will be useful. Rather than experimenting and measuring things that do not exist, we can construct models of the competing

schemes and predict what the cost/performance comparisons will be. Then, as implementations become available for measurement, we can begin to validate the modeling approach and test the predictions. At this early stage of distributed database experience, such modeling and the analysis which leads to prediction seems to be quite an appropriate alternative.

4.2 A MODELING APPROACH

Response time is defined as the interval between a user's request for a data transaction (retrieval or update) and his notification of its completion. In a distributed system, "completion" may mean a variety of things. For example, completion of an update in a master/slave system could mean the update has been applied at the master copy and saved for later transmission to slave copies. In contrast, a synchronized system might consider an update complete only when all copies have acknowledged application. Interactive retrieval completion is simpler, because the user receives his result at his terminal. To encompass some of this variety, we will consider a transaction to be complete when the user receives notice that he can proceed with his next transaction.

We separate contributions to response time (RT) into the following stages (explained below):

* process request (P1)

* optimization strategy (OS)
* transmission to remote nodes (CD1)
* process data (PD)
* transmission from remote nodes (CD2)
* process result (P2).

For an individual data transaction, the response time will be

$$RT = P1 + OS + f(CD1, PD, CD2) + P2,$$

where the function f depends on the network topology and on the actual management scheme used.

The process request stage interprets the transaction and maps it from the user's view of the limited database portions he can access to a global (logical) database view. If the request is complicated, it is usually decomposed into a hierarchy of simple steps. INGRES, for example (see section A.7.1 of the appendices), decomposes multi-variable queries to allow multiple one-variable processing. In a centralized system, the one-variable steps would be done sequentially; in a distributed system they could be done in parallel at multiple nodes if the data were distributed.

In the optimization strategy stage, the request is distributed to the nodes holding appropriate pieces of the data or any special programs required. OS will represent strictly the processing cost at the originating node. This can be very complicated if selection of some combination of data movements and local processing is desired (as in SDD-1,

see section A.7.2 of the appendices). The optimization can be time-consuming and the appropriateness of particular algorithms may not be established until implementations demonstrate the performance capabilities. We will not deal with optimization strategy at all in this analysis because it would involve modeling at a level of detail much lower than would be productive for the simple, average cost/performance indicators we chose.

Transmission to remote nodes, if any is required, involves setting up all of the logical communications links required, actually transferring the data and any waits for acknowledgements.

The process data stage begins with mapping from the global data view to appropriate local physical storage requests. Disk accesses will get or put data for retrievals and updates, and any results to be returned will be mapped back to the global data view. These activities may occur in parallel at different nodes if more than one holds data necessary to a transaction.

Transmission from remote nodes returns completion information or results. Set-up time here may be minimal if the sending logical connection remains established and open as a virtual communications circuit.

The process result stage collects and coordinates any remote results being returned, transforms from the global data view back to the local user's view, and does any processing required to present results to the user.

The equation for response time is designed to emphasize the effects of distribution on response time. It will be used to evaluate the proposals of this dissertation to trade off timeliness constraints and to delay synchronization, since either of these will contribute to the particular nature of the function f.

In order to concentrate on comparing the management schemes, we will take a very narrow viewpoint and reduce our consideration to just the major contributions of processing data (PD) and transmission to and from remote nodes (CD1, CD2):

$$RT = f\ (\ CD1,\ PD,\ CD2\ )\ .$$

We consider this to be the first-order statistics of the problem, and consequently look to first-order queuing analysis for the solutions. That is, the average contribution from (P1 + OS + P2) will be so close to constant for the various management schemes, that it will contribute only a constant offset in the average RT value. Second-order statistics, such as the variance, are probably not so simply partitioned and will not be dealt with here. In this chapter, we consider, then, Poisson arrivals and exponentially distributed service

times with a single server at each node (central or distributed), the basic M/M/1 queuing situation [KLEI75]. A summary of the notation used is provided as Table 4-1.

In order to focus on the distribution issues of the various management schemes, we will use a further simplification of constant, fixed communication delays between terminals and a central site or between any pair of nodes in a network supporting distributed management. Thus, CD1 = CD2 = CD , a constant. Since this generally represents an idealized, best-case communications delay, we will actually be computing lower bounds for the true values of response time. This is quite appropriate, since our primary interest is in comparing the management schemes, not in predicting actual performance values for implemented systems. Choice of some maximum allowable communications delay that included a tolerance for resource contention and addition of some maximum offset value for (P1 + OS + P2) would be needed in order to calculate approximate upper bounds instead.

Update and retrieval transactions will be assumed to arrive uniformly distributed over all the terminals in a system, and arrival rates will be expressed per terminal. Complete duplicate copies of the database will be assumed at each node except under delsync management, where the copies naturally diverge between merges.

Table 4-1. Summary of Notation, Basic

CD	communications delay
L	average system arrival rate
LR	average arrival rate per terminal of retrieval transactions
LU	average arrival rate per terminal of update transactions
XBAR	average system service time
XR	average service time for retrievals
XU	average service time for updates
n	number of terminals per node
N	number of nodes
rho	utilization
T	average total time spent in the system
RT	response time
A=LU/LR	update/retrieval ratio per terminal

To evaluate the QR-BI and delayed synchronization proposals of this thesis, we will wish to extend the basic models to encompass explicitly the data vs. access timeliness tradeoff and add a model for the delayed synchronization management scheme. The approach is the same as in the basic analysis:

* describe the system flow of transactions according to

each management philosophy,

* use queuing theory (or the operational method) to compute the average response time as a function of the system parameters, and
* pick a scenario or standard set of parameters to use as a basis for comparison.

The queuing model that is appropriate for the QR-BI environment is the head-of-the-line, non-preemptive, priority queue [KLEI76]. We will use two priority classes: high for QR retrievals and low for BI retrievals and for updates. Within a priority class, order is first-come-first-served on an arrival basis. This actually introduces a new, extra dimension of "quickness" to QR, since up to this point we have been treating it strictly as a function of locally or remotely stored data.

To compute average response time, average total times for the different transaction types in each node are combined with appropriate communications delays according to the particular management schemes. The basic assumptions of constant, unit communications delay, constant system load of 100 terminals in the system, and uniform distribution of transaction arrivals are carried over into the QR-BI scenario from the basic case.

From Kleinrock's [KLEI76] derivations for head-of-the-line priority queuing, we specialize to two priority classes and exponential service distributions to get

average queuing wait times (see Appendix B). New notation is defined in Table 4-2; for previously used symbols, refer back to Table 4-1.

Table 4-2. Summary of Notation, Extended

W0	average wait due to transaction in service
LQR	arrival rate per terminal of quick response retrieval requests
LBI	arrival rate per terminal of best information retrieval requests
B	ratio of QR/BI arrivals
L1	arrival rate of low priority transactions to the priority queue
L2	arrival rate of high priority transactions to the priority queue
X1BAR	average service time for low priority transactions
X2BAR	average service time for high priority transactions
W1	average wait time (in queue) of low priority transactions
W2	average wait time (in queue) of high priority transactions

sigma1 = L1 * X1BAR + L2 * X2BAR

sigma2 = L2 * X2BAR

TQR	average total time (in queue and in service) for QR retrieval
TBI	average total time (in queue and in service) for BI retrieval
TU	average total time (in queue and in service) for update
p(local)	probability that a retrieval can be satisfied from local data

4.3 THE MODELS AND THE ANALYSIS

4.3.1 Centralized Management

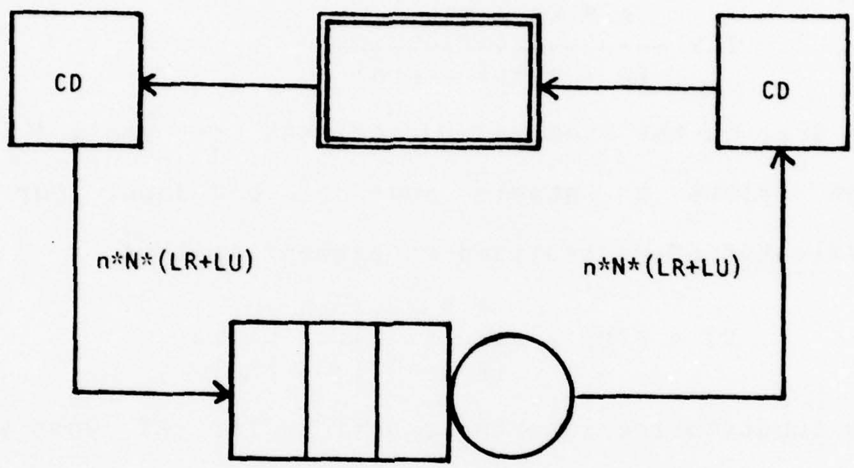
A centralized database system consists of a central site receiving update and retrieval transactions from all the terminals in the system. We assume that the terminals are located a fixed communication delay, CD, from the central site. The flow diagram for the queuing model of a centralized system is shown in Figure 4-1. Its purpose is to explicitly show the arrival rates, the communication paths and delays, and the logical flow of the transactions through the system (e.g., transactions arrive from and are returned to the cluster of terminals, which is represented by the double rectangle in the diagram).

The mathematical details of the queuing analysis for the model are in section B.1.1 of the appendices. The average response time for the system can be expressed by

$$RT = CD1 + T + CD2,$$

where CD1 is the communications delay between the terminals and the central node, T is the total time spent by a transaction in the queuing system, and CD2 is the communications delay from the central node back to the terminals. The total time in the queuing system is a function of the service times for updates (XU) and retrievals (XR), the ratio of update arrivals to retrieval arrivals (A), and the

Figure 4-1. Flow Diagram for Basic Centralized System



utilization of the server (rho, which can also be thought of as what fraction of time the central node is busy processing the transactions):

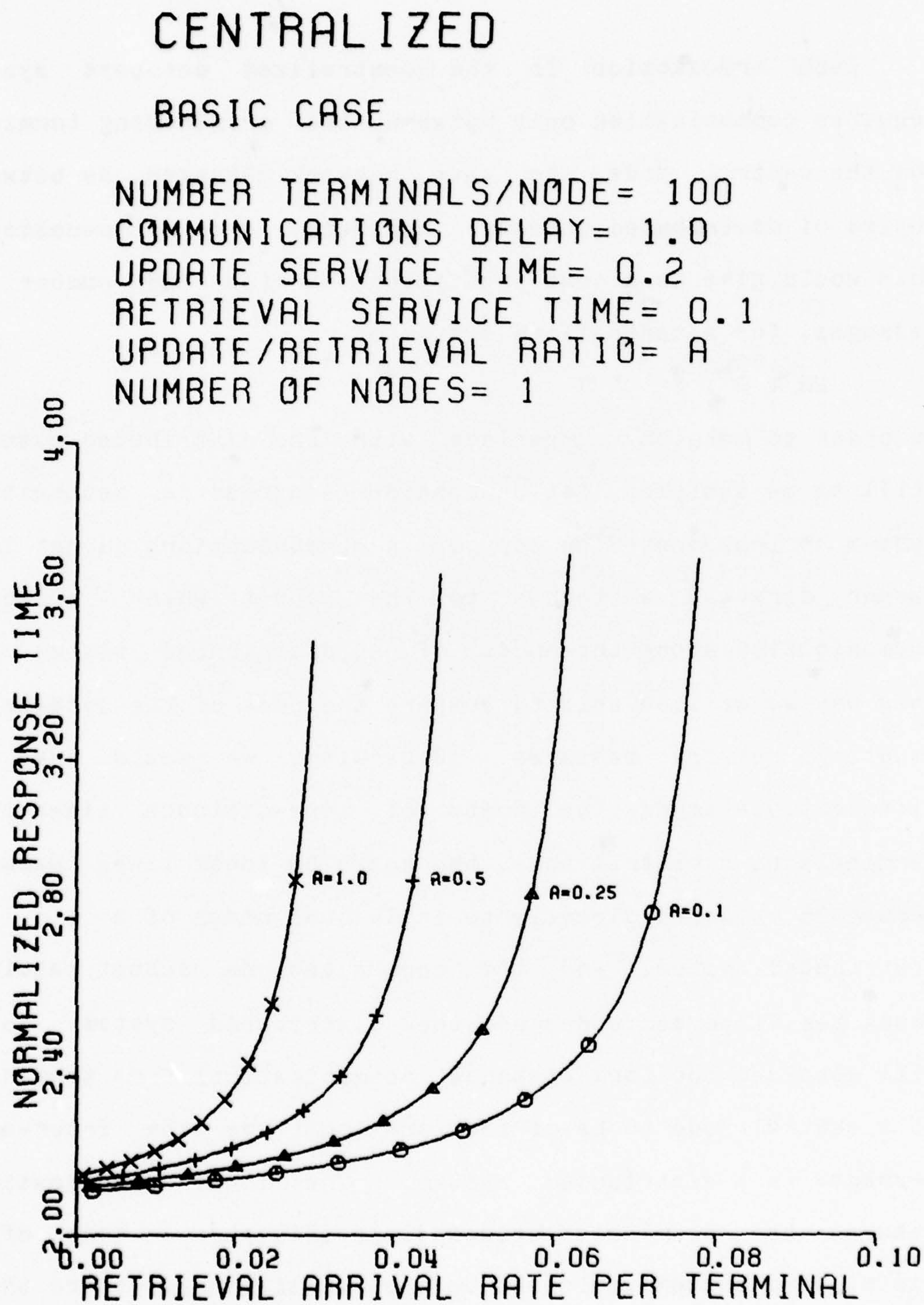
$$T = \frac{A * XU + XR}{(A + 1)*(1 - rho)} \quad .$$

We are, on the average, restricted to rho < 1 in order that the system be stable and not bog down. Our performance evaluator of centralized management is thus

$$RT = 2*CD + \frac{A * XU + XR}{(A + 1)*(1 - rho)} \quad ,$$

by substituting into the equation for RT what we know about CD1, CD2, and T. Figure 4-2 shows how this response time varies with the update/retrieval ratio. The numerical values have been chosen on the basis of a unit communications delay, so that if we consider a communications delay of 1 second (reasonable from section 3.1.3), we are talking about update service time of 200 ms, and retrieval service time of 100 ms (reasonable for large database disks commonly used today). Retrieval arrival rate is chosen as the independent variable for computational convenience and because the system performance does depend on the arrivals very basically (without transaction arrivals the system has no work to do and performance is meaningless). On this basis, response time comes out in seconds.

Figure 4-2. Dependence of Response Time on the Update/Retrieval Ratio



Each transaction in the centralized database system requires communication only between the originating terminal and the central node, so that network messages, as between copies of distributed database managers, are not necessary. This would give us a cost prediction, NM for the number of messages, for a centralized system of

$$NM = 0 .$$

In order to make any comparison with the distributed systems still to be analyzed, let us consider instead a centralized system as implemented on top of a communications subnet in a manner directly analogous to the subnet which supports communication among the nodes of a distributed system. In this way we will be able to compare the cost of the systems by counting network messages. Otherwise, we would have to differentiate among the costs of long-distance lines from terminals to a central node, the costs of local lines between terminals attached directly to individual nodes of a distributed system, and the communications subnet required among the dispersed nodes of the distributed system. So we will consider the long-distance communications from terminals to a central node to be of the same cost as the inter-node messages in a distributed system. The local communications between the terminals attached locally to a node of a distributed system would not be counted similarly, since their contribution to the operational cost of the distributed system is not comparable. For this analysis, then, the cost

stages (explained below):

* process request (P1)

predictor of a centralized system is

$$NM = 2 .$$

System flow for centralized management of a QR-BI database system is summarized in Figure 4-3. The two types of retrievals and priority queuing are shown explicitly. The total processing load is the same as in the basic case, and it seems likely that reordering the queue will have little effect on the <u>average</u> system response time (see the discussion of conservation laws in [KLEI76] for more detail), although it may greatly influence the variance, a second order statistic we have not been considering. In Figure 4-4, the B=0 case where there are no QR requests at all is exactly the basic case. Indeed, the introduction of QR and BI retrievals does not change the response time. Figures 4-5 and 4-6 show how the contributions to the average shift as the ratio of QR/BI changes. The dependence of the response time on the update/retrieval ratio is shown in Figure 4-7.

can be very complicated if selection of some combination of data movements and local processing is desired (as in SDD-1,

Figure 4-3. Flow Diagram for Centralized QR-BI System

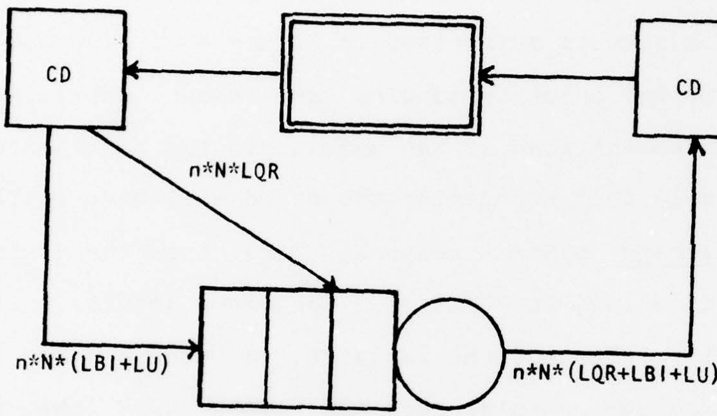


Figure 4-4. Dependence of Response Time on the QR/BI Ratio

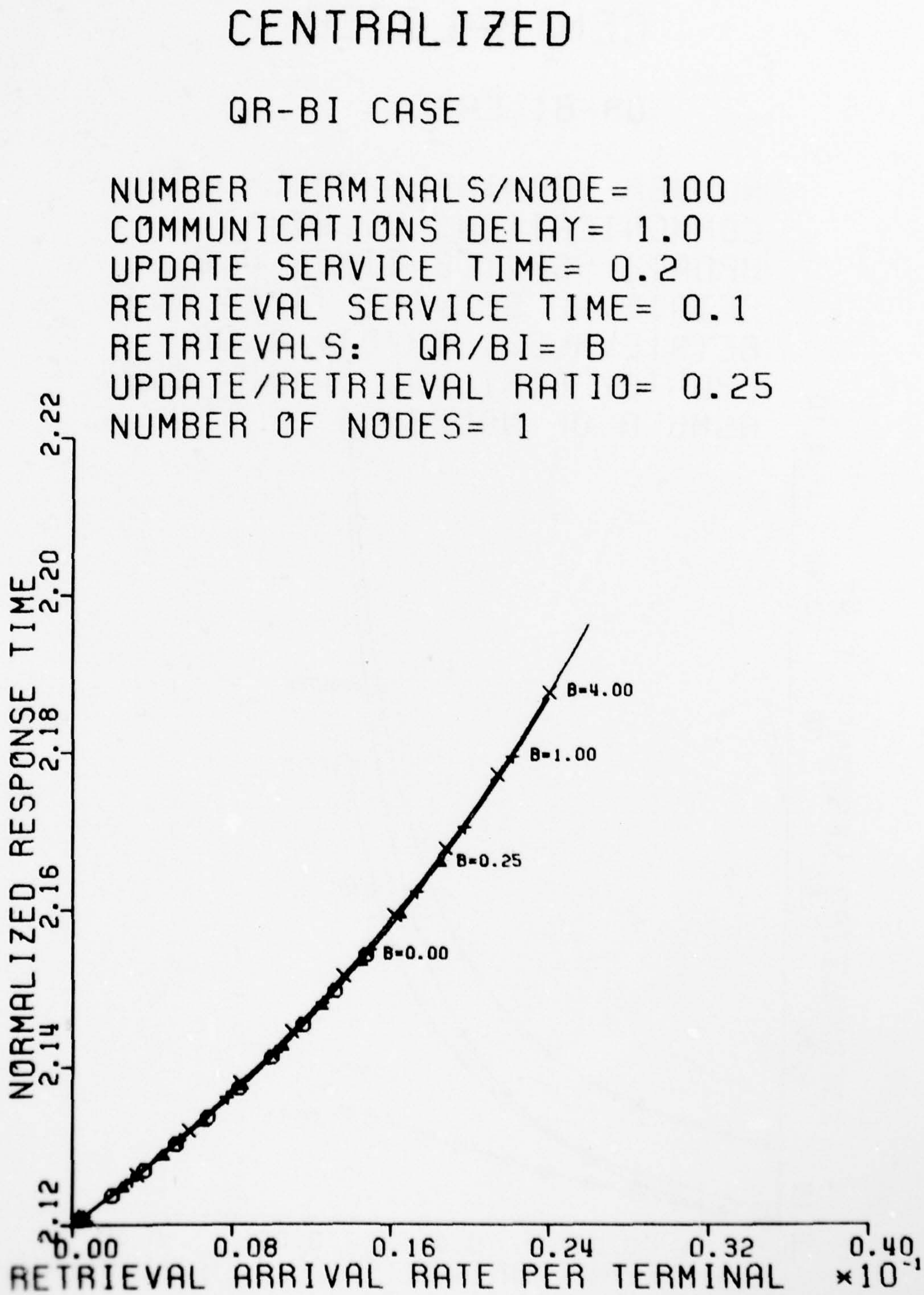


Figure 4-5. Average Response Time from Contributions for QR/BI=1

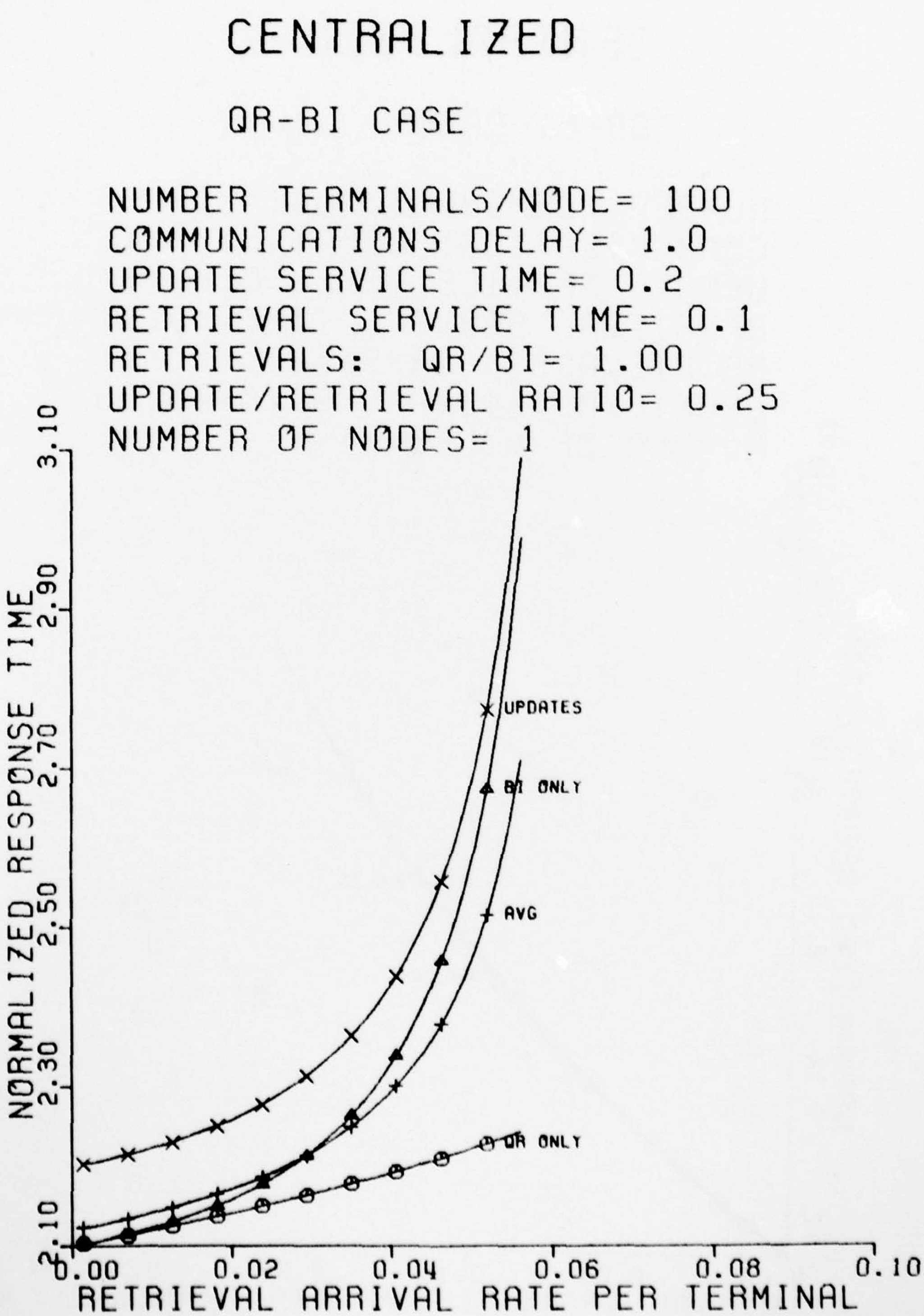


Figure 4-6. Average Response Time from Contributions for QR/BI=4

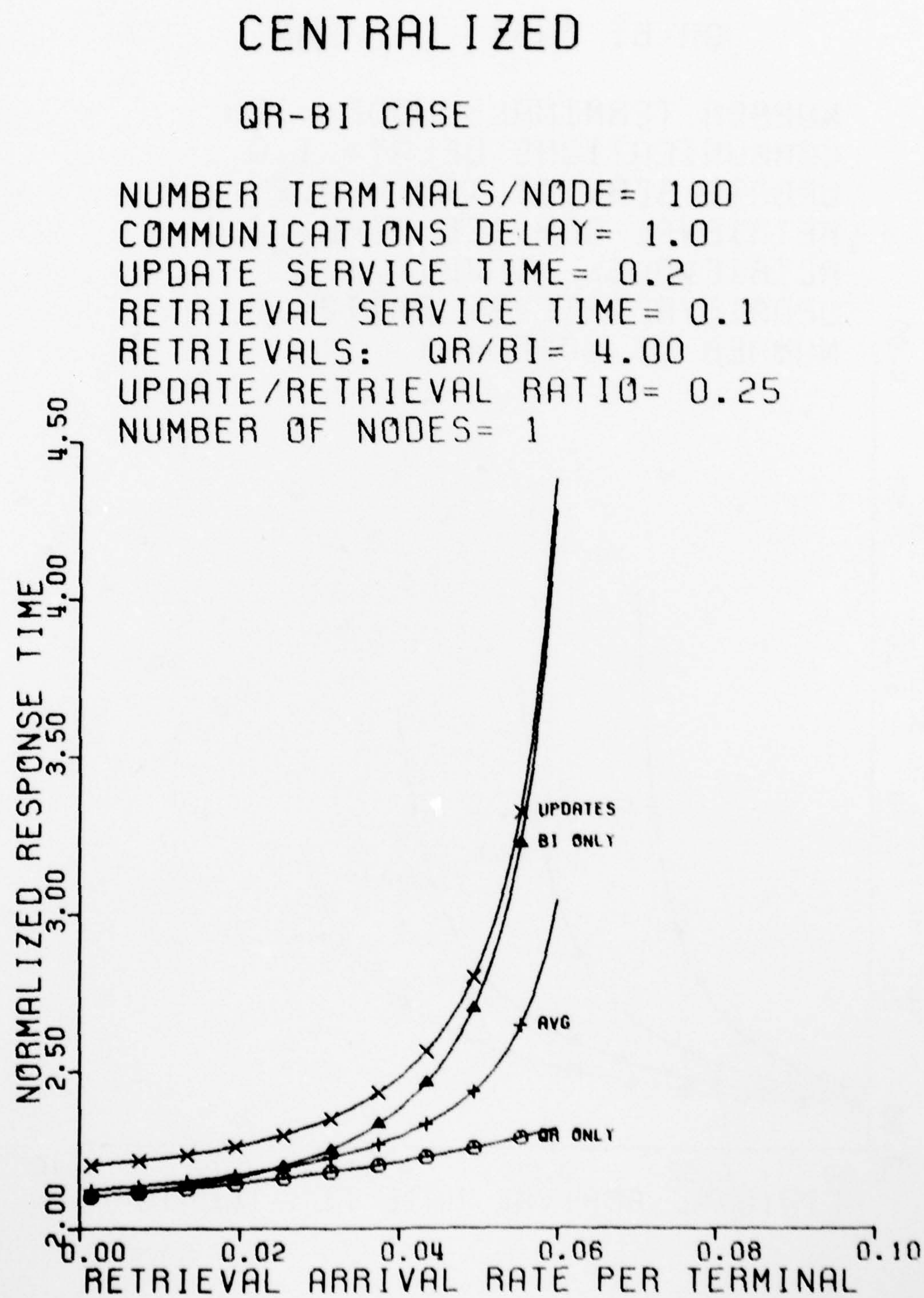
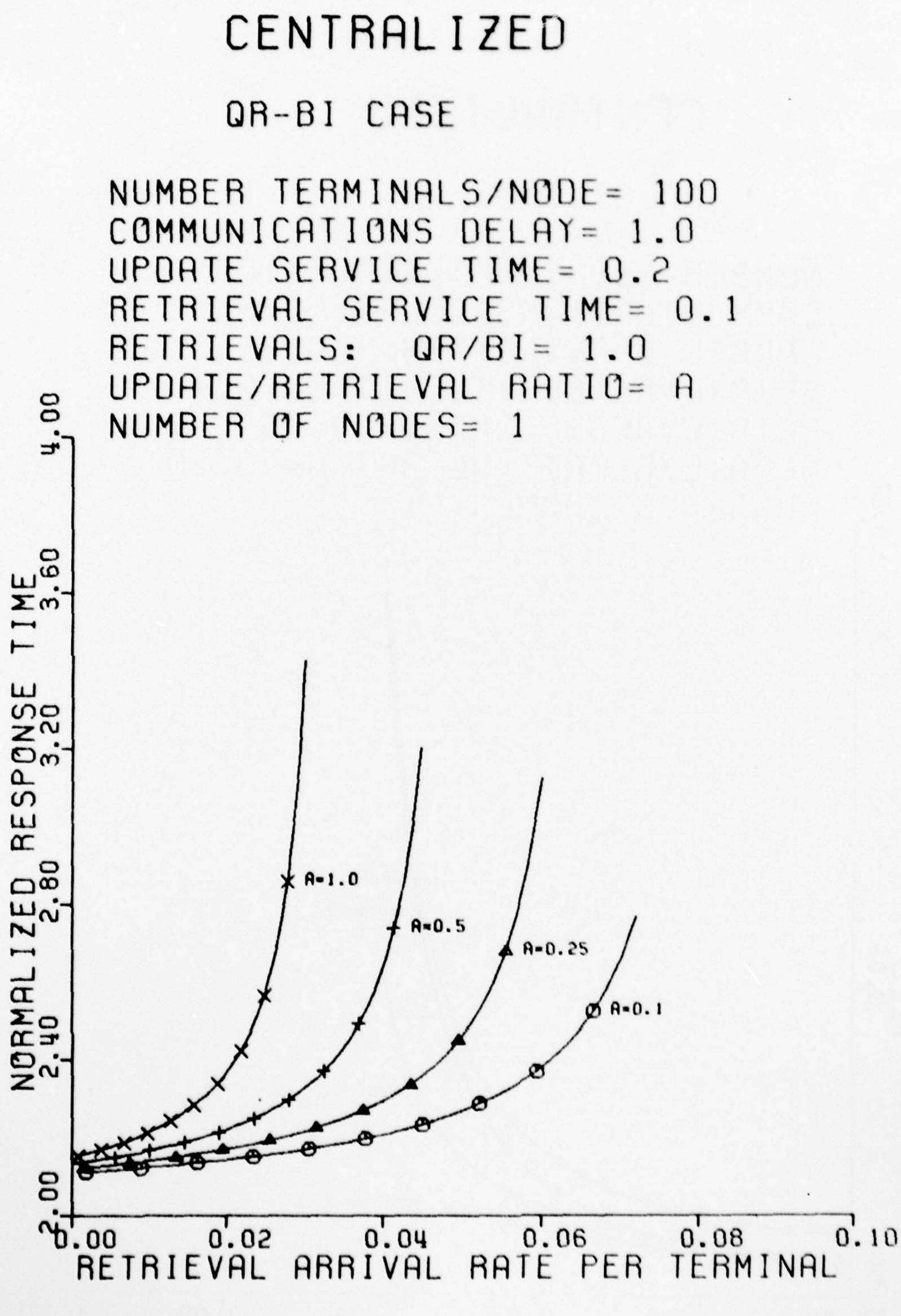


Figure 4-7. Response Time Dependence on the Update/Retrieval Ratio



4.3.2 Master/Slave Management

We will consider two-host resiliency (see section A.2.2.2.3 of appendices) for the master/slave management of a distributed database system: all updates are forwarded to the master for application, then to the back-up, and finally out to all the slaves. The acknowledgement to the user of the system's acceptance of the update transaction is sent only after the back-up has successfully applied the update to its (complete duplicate) copy of the database. Each node will process all retrieval requests from its own local terminals against its own (complete duplicate) copy of the database. The network flow diagram for the queuing model of the system is shown in Figure 4-8, and the flow internal to the nodes is in Figure 4-9. By adding up flow in and out of each node, we see that the processing traffic required is the same for each node in the network (see section B.1.2 of the appendices for the details). This gives a total time in the queuing system of

$$T = \frac{N * A * XU + XR}{(N * A + 1)*(1 - rho)} \quad .$$

The average response time throughout the network must take into account both the different types of nodes and the different transactions. For the one master, one back-up, N-2

Figure 4-8. Network Flow Diagram for Basic Master/Slave System

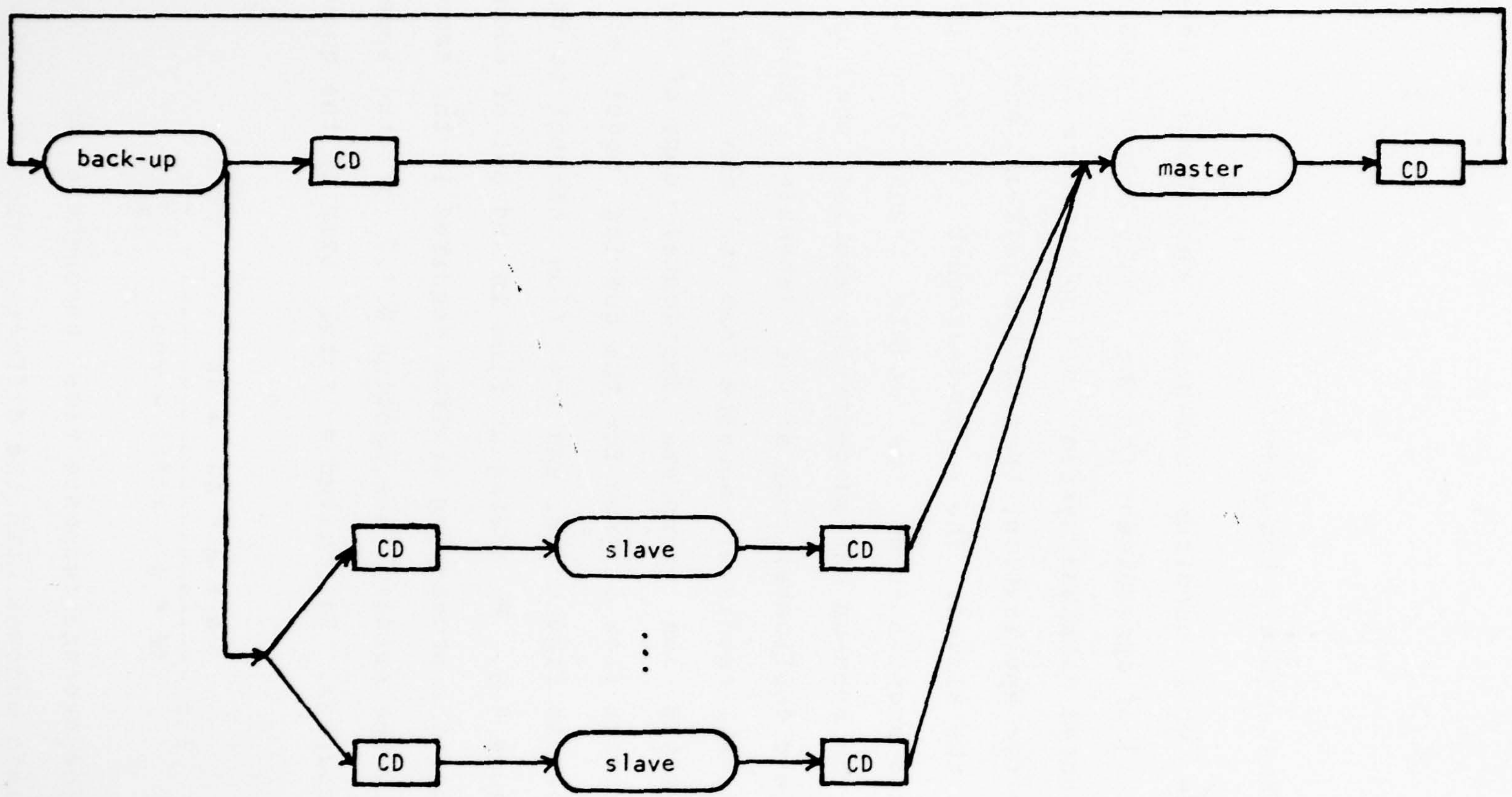
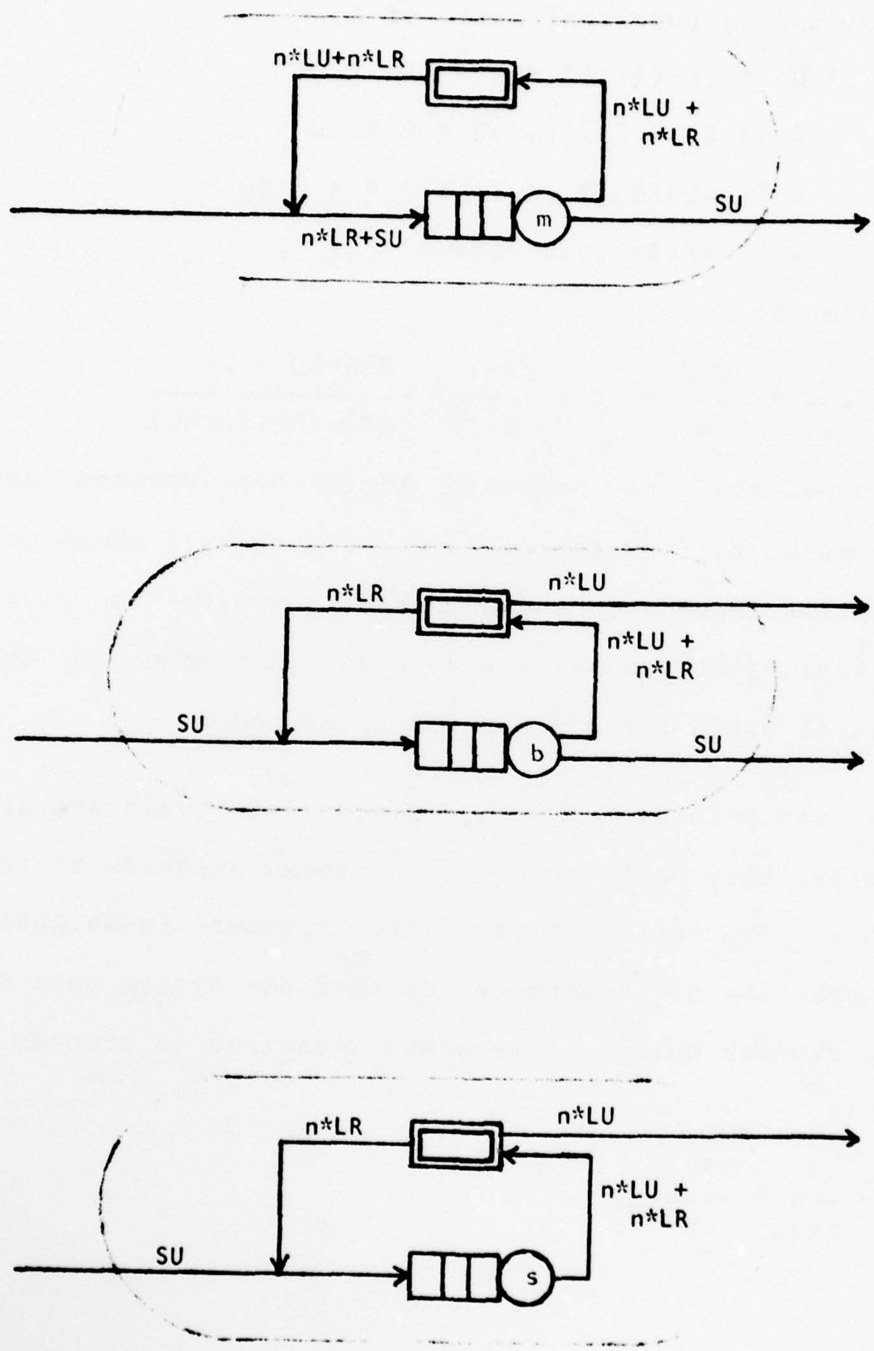


Figure 4-9. Internal Node Flow for Basic Master/Slave



slaves, and an update/retrieval ratio of A,

$$RT = (1/N) * [A/(A+1)] * 2 * CD$$
$$+ (1/N) * [A/(A+1)] * 2 * CD$$
$$+ [(N-2)/N] * [A/(A+1)] * 3 * CD$$
$$+ [2*A/(A+1) + 1/(A+1)] * T \quad ,$$

which simplifies to

$$RT = \frac{A}{A+1} * \frac{3N-2}{N} * CD + \frac{2A+1}{A+1} * \frac{N*A*XU + XR}{(N*A+1)*(1-rho)} \quad .$$

Figure 4-10 shows how the response times for updates and retrievals combine to form the average. Figure 4-11 shows how the response time depends on the number of nodes for a given update/retrieval ratio, and Figure 4-12 how it depends on the update/retrieval ratio for a given number of nodes.

In the master/slave system, since retrievals are all handled locally, they contribute no network messages to the cost predictor. The contribution of the updates is weighted by their probability of occurrence, so that the system cost in terms of the average number of messages required to process a transaction is

$$NM = \frac{A}{A+1} * \frac{3*N-2}{N} \quad .$$

Figure 4-10. Average Response Time from Contributions

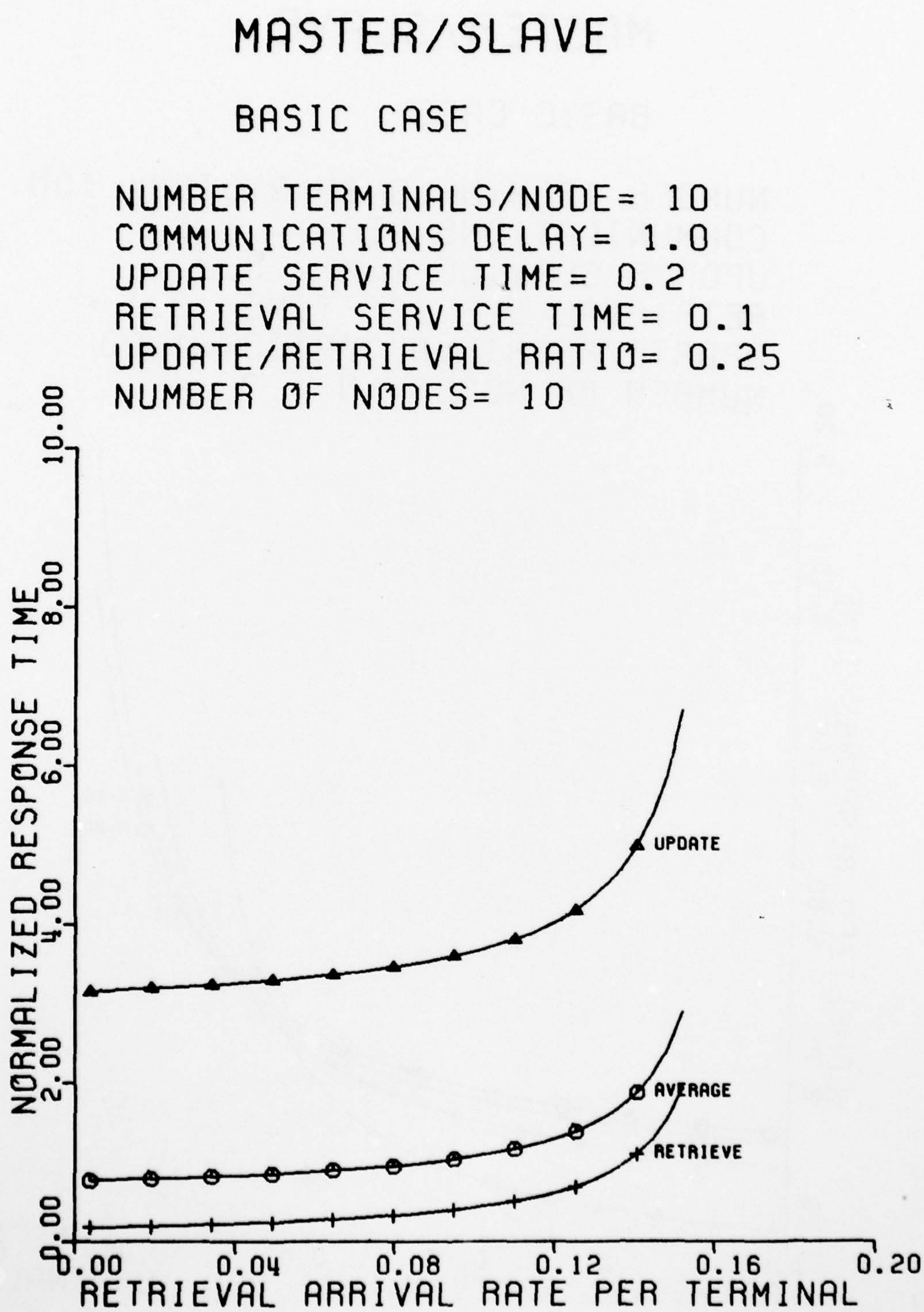


Figure 4-11. Dependence of Response Time on the Number of Nodes in the System

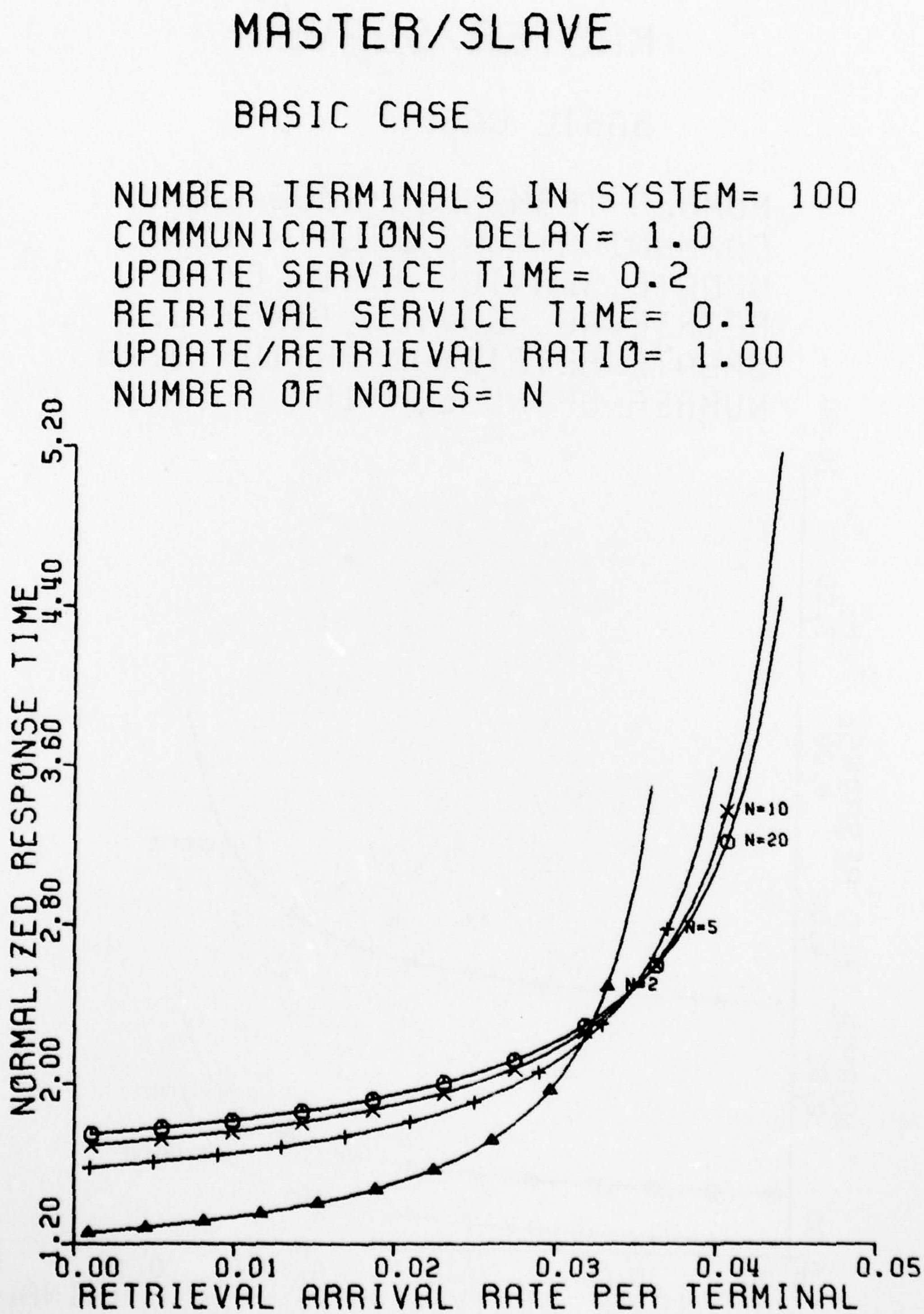
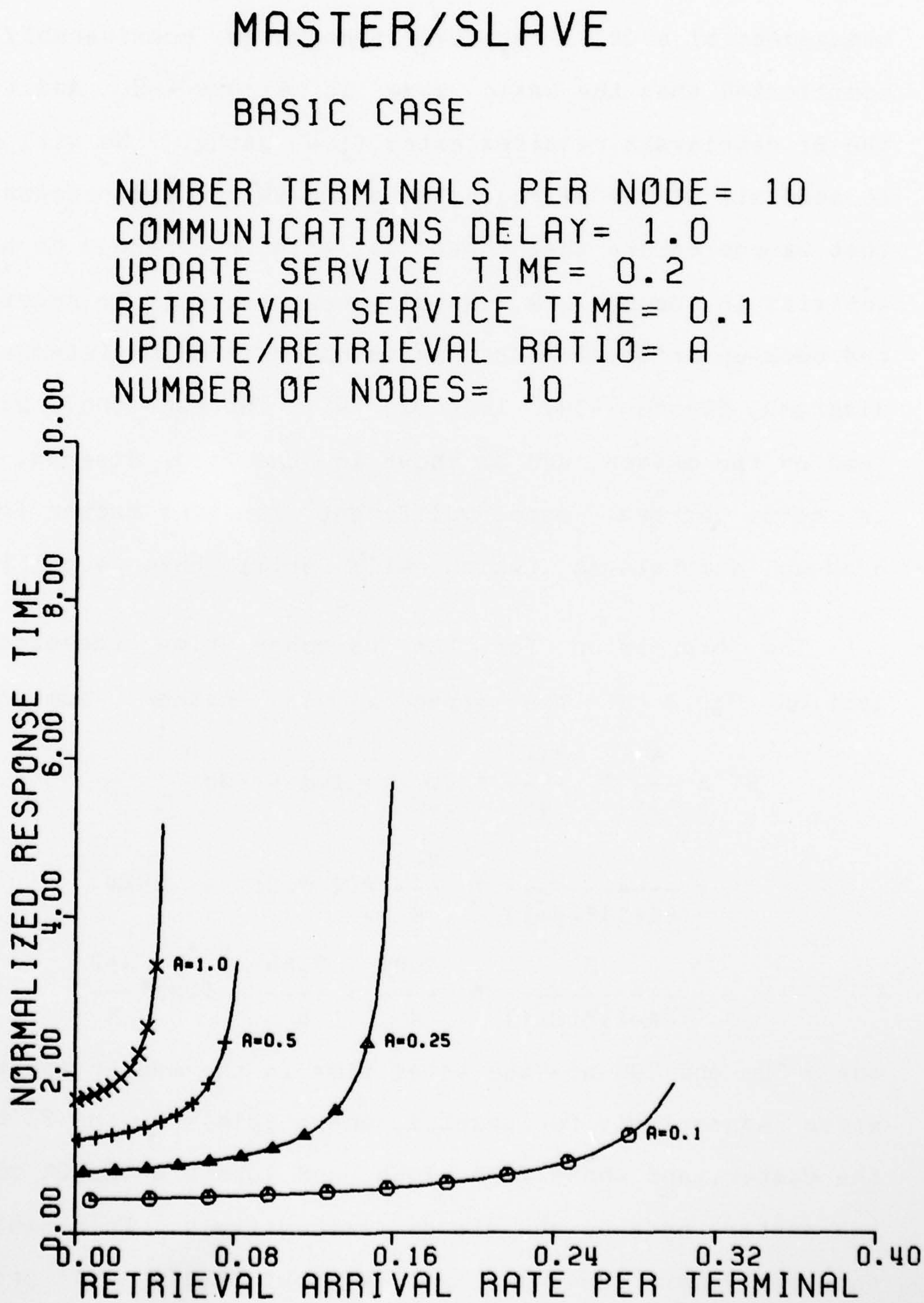


Figure 4-12. Response Time Dependence on the Update/Retrieval Ratio



The system flow in Figure 4-13 for master/slave management of a QR-BI retrieval system is considerably more complicated than the basic case in Figure 4-8. Addition of the BI retrievals requires extra flow paths. We will choose to send all of the BI requests to the master for processing so that we can ensure that "best" is with reference to all the activity in the network. No BI processing will be required in the back-up or slave nodes, as is clear from the internal flow diagrams, Figure 4-14. The BIs will increase the processing load on the master, and as shown in the flow diagram, result in total arrival rates different for the master from the back-up and slaves (which will still have equal loads).

The expression for the response time (developed in section B.2.2 of the appendix) is rather cumbersome:

$$RT = \frac{A}{A+1} * \frac{3N-2}{N} * CD + TUm + TUb$$

$$+ \frac{1}{(A+1)*(B+1)} * \frac{N-1}{N} * 2 * CD + TBIm$$

$$+ \frac{B}{(A+1)*(B+1)} * \frac{TQRm}{N} + \frac{TQRb}{N} + TQRs*\frac{N-2}{N}$$

where TUm and TUb are the total time in the master and back-up nodes respectively for updates, where TBIm is the BI time in the master, and where TQRm, TQRb, and TQRs are the QR times in the master, back-up and slaves, respectively. This additional complexity over the basic case (without QR-BI) will prompt us to look for significant performance or cost benefits in order

Figure 4-13. Network Flow Diagram for Master/Slave QR-BI System

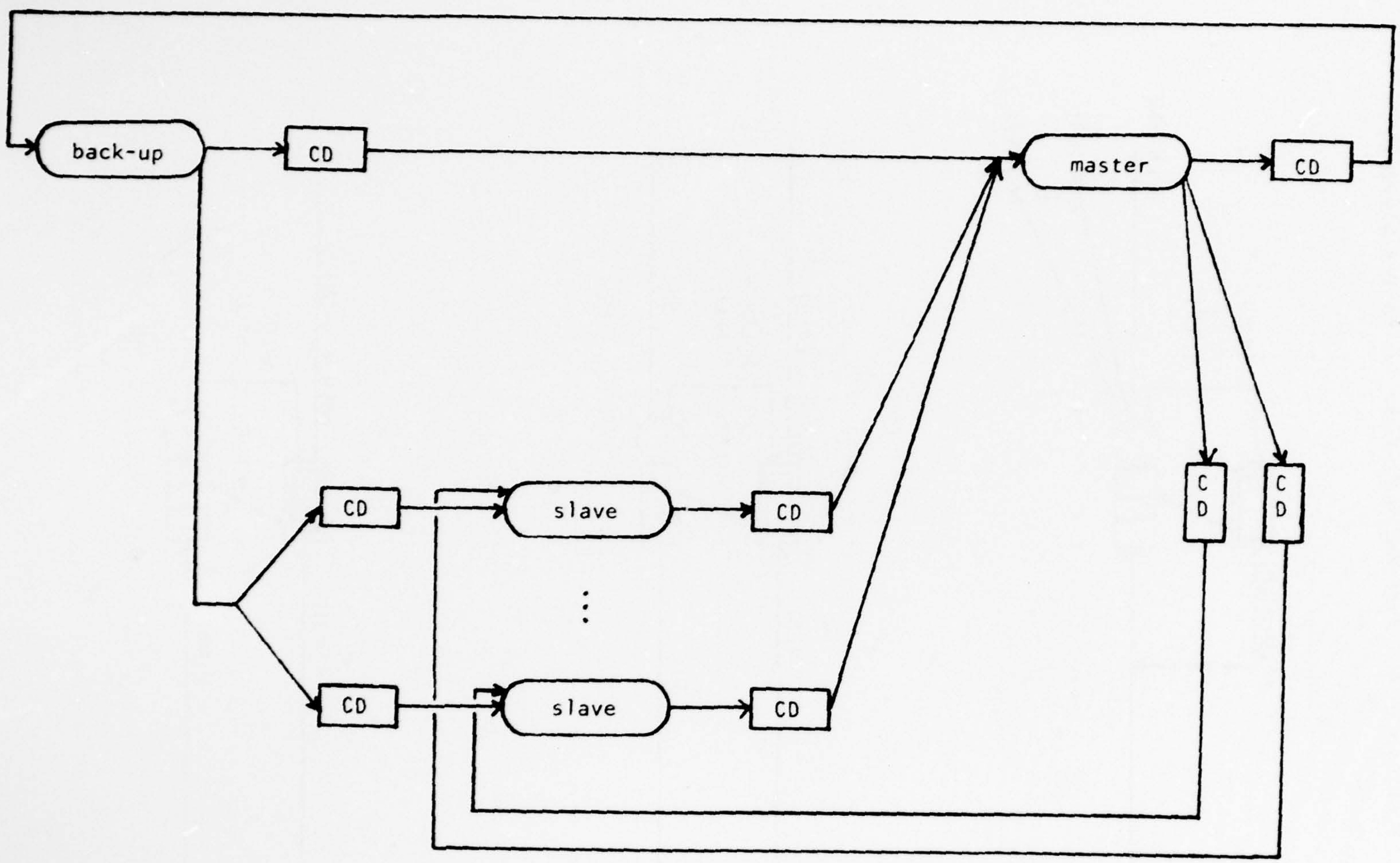
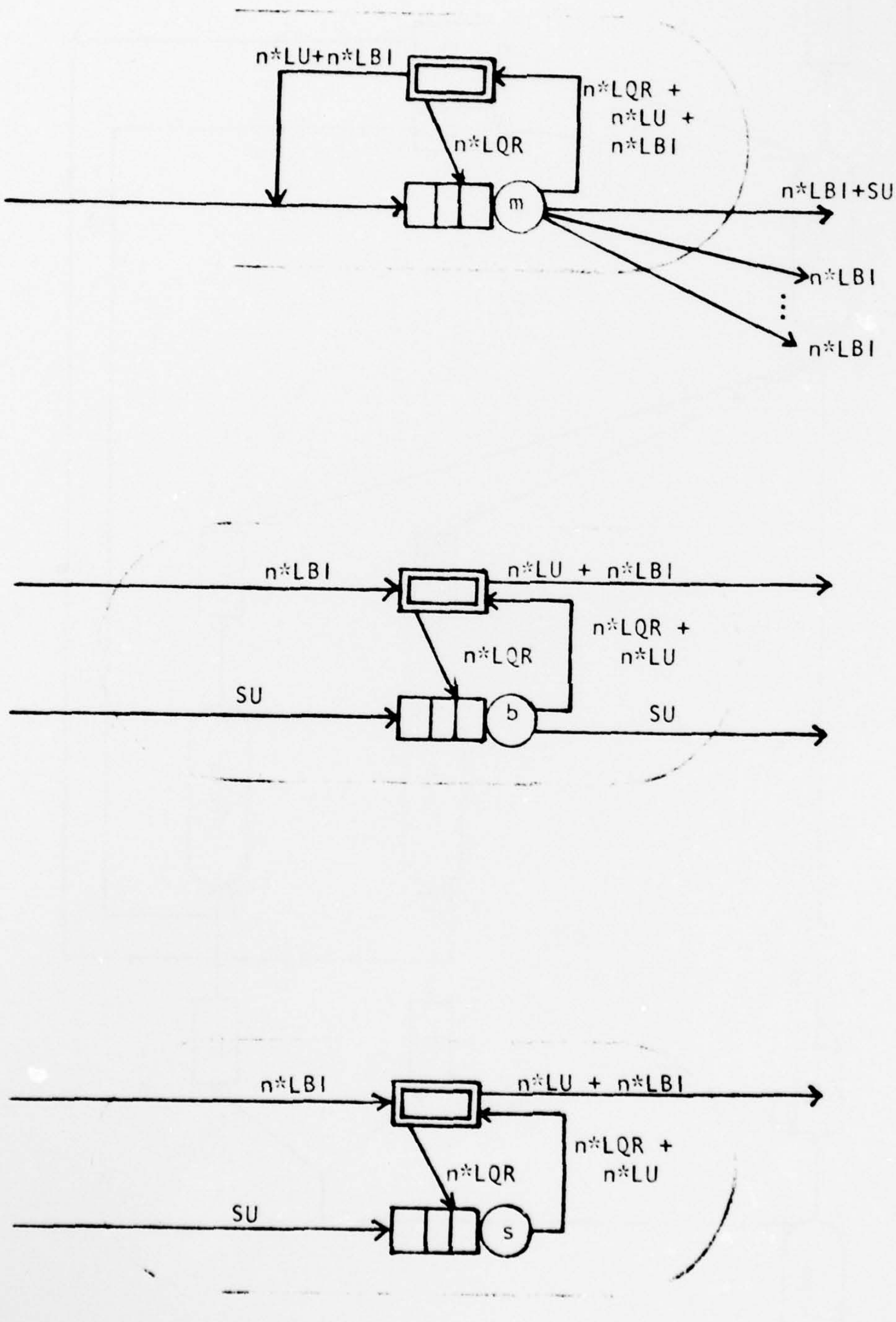
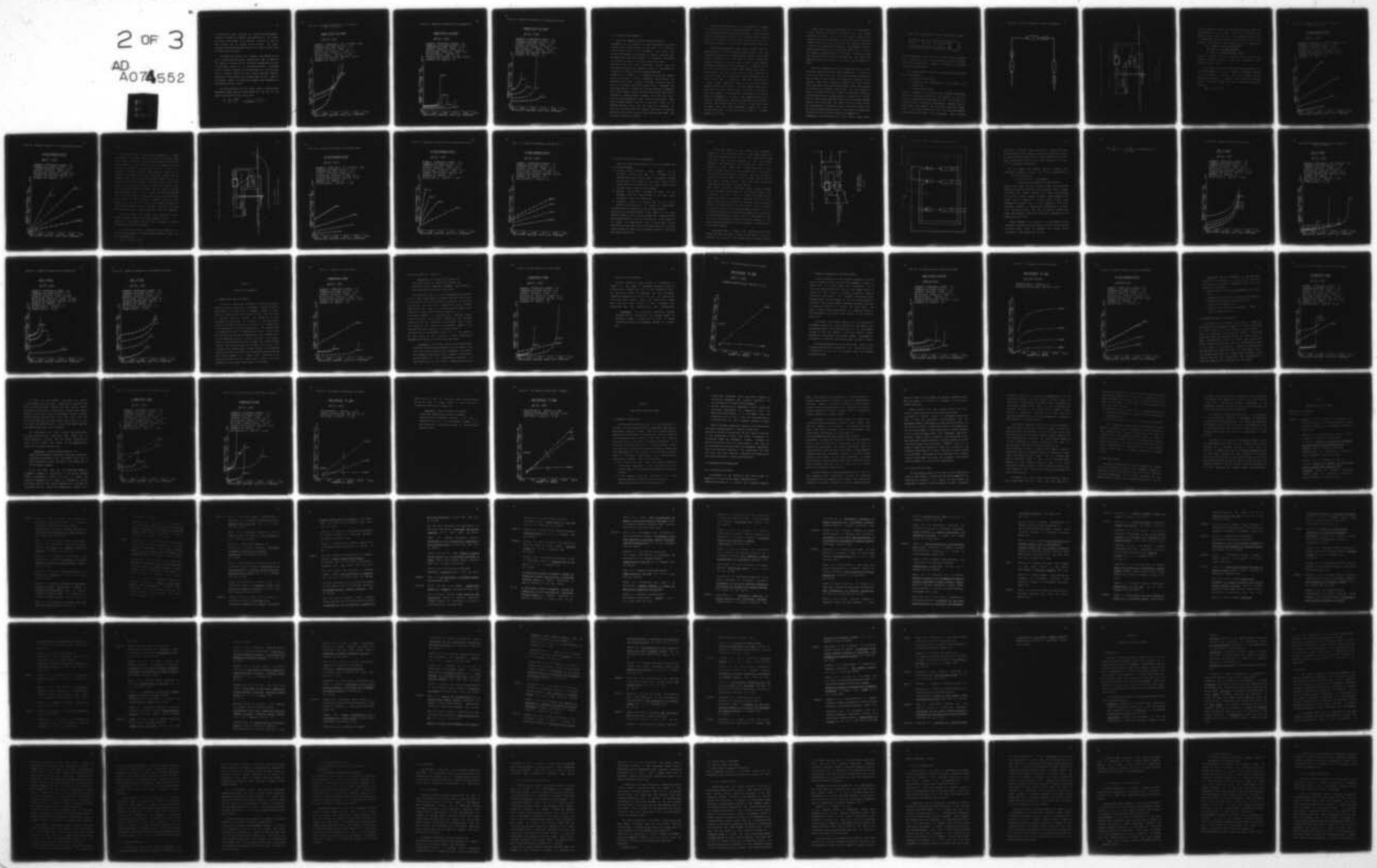


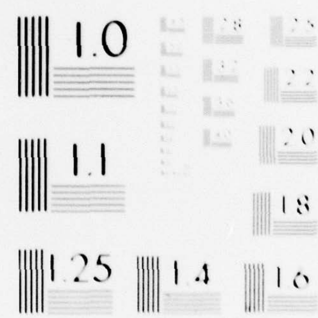
Figure 4-14. Internal Node Flow for Master/Slave QR-BI



AD-A074 552 NAVAL UNDERWATER SYSTEMS CENTER NEW LONDON CT NEW LO--ETC F/G 5/2
PERFORMANCE AND TIMELINESS IN A DATABASE.(U)
JUL 79 L A DENOIA
UNCLASSIFIED NUSC-TR-6099 NL

2 OF 3
AD
A074552





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

to justify the QR-BI approach for master/slave management. Figure 4-15 shows the response time dependence on the number of nodes in the system. It is similar to Figure 4-11 for the basic model, but we already notice trouble: the master saturates and drives response time up at lower arrival rates for QR-BI than for basic.

In Figure 4-16 we see how response time depends on the ratio of QR and BI retrievals. Notice here that a QR/BI of zero means all update and (BI) retrieval requests must be sent to the master, much as in a centralized system. Increasing values of QR/BI represent increasing amounts of activity that can be handled locally by the slaves, reducing resource contention at the master and allowing higher arrival rates to be handled. Figure 4-17 shows the response time dependence on the update/retrieval ratio.

The cost prediction for the QR-BI model of master/slave management simply adds the contribution for the BIs to the update contribution of the basic case:

$$NM = \frac{A}{A+1} * \frac{3N-2}{N} + \frac{1}{(A+1)*(B+1)} * \frac{N-1}{N} * 2 \quad .$$

Figure 4-15. Dependence of Response Time on the Number of Nodes in the System

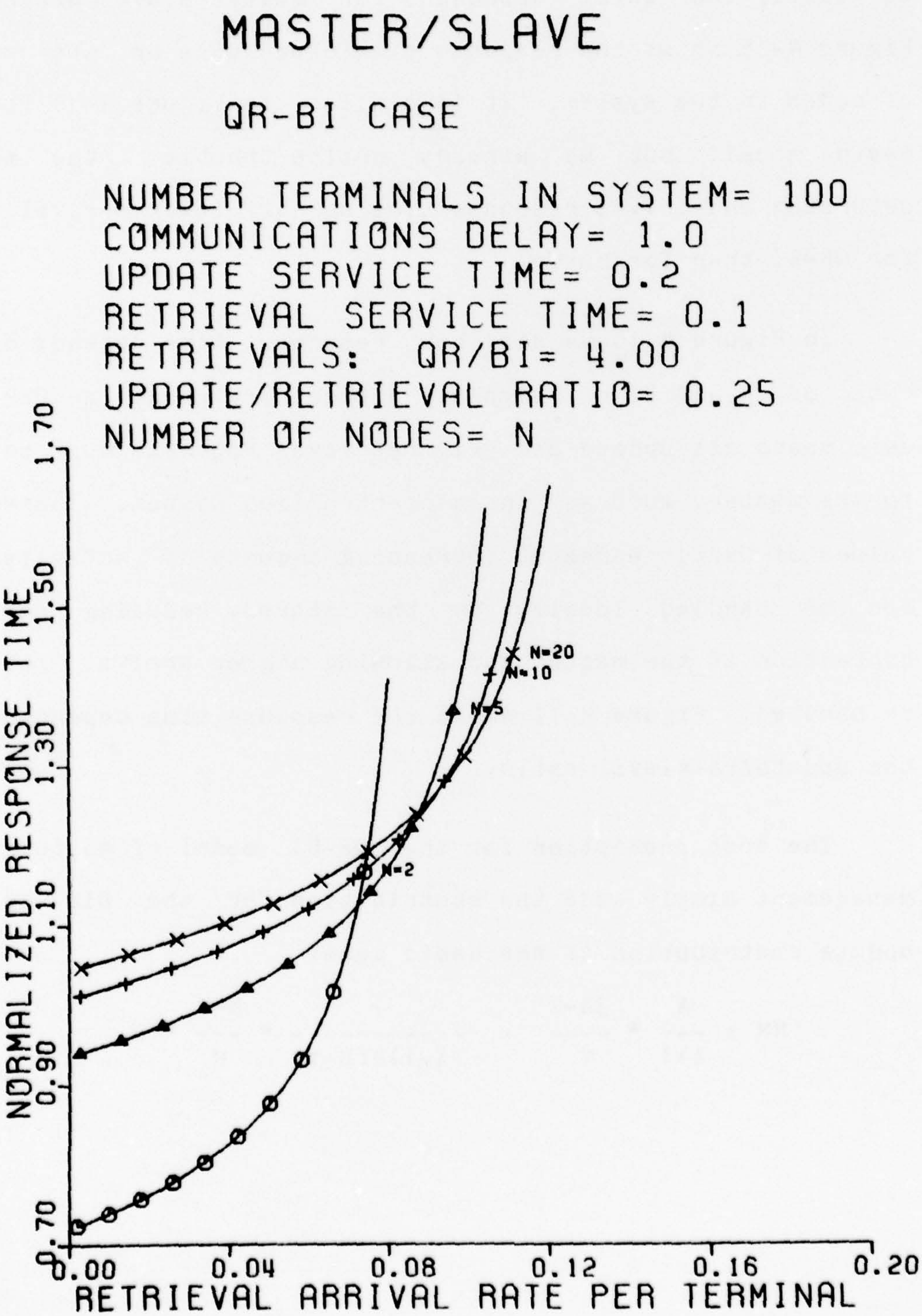


Figure 4-16. Dependence of Response Time on the QR/BI Ratio

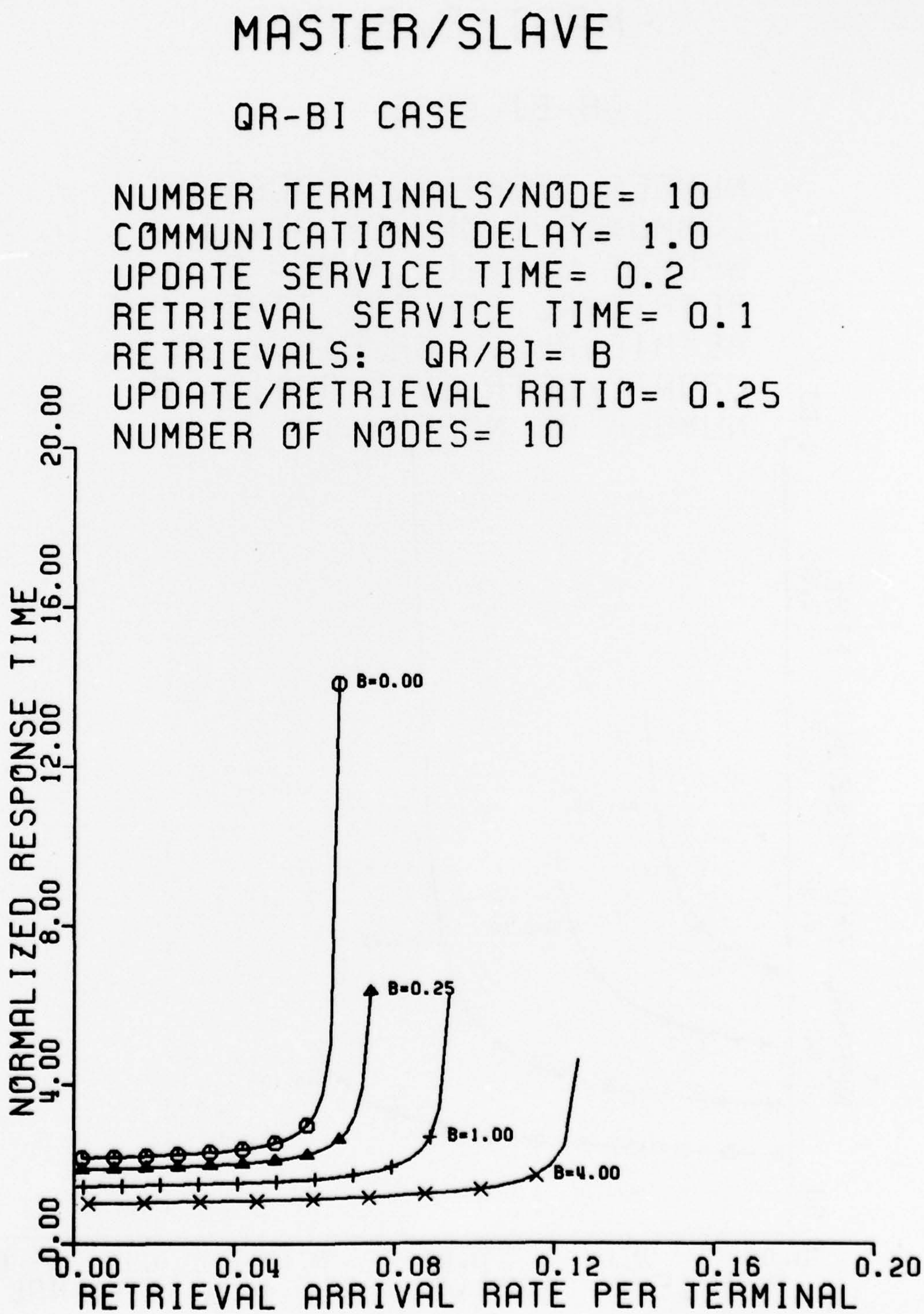
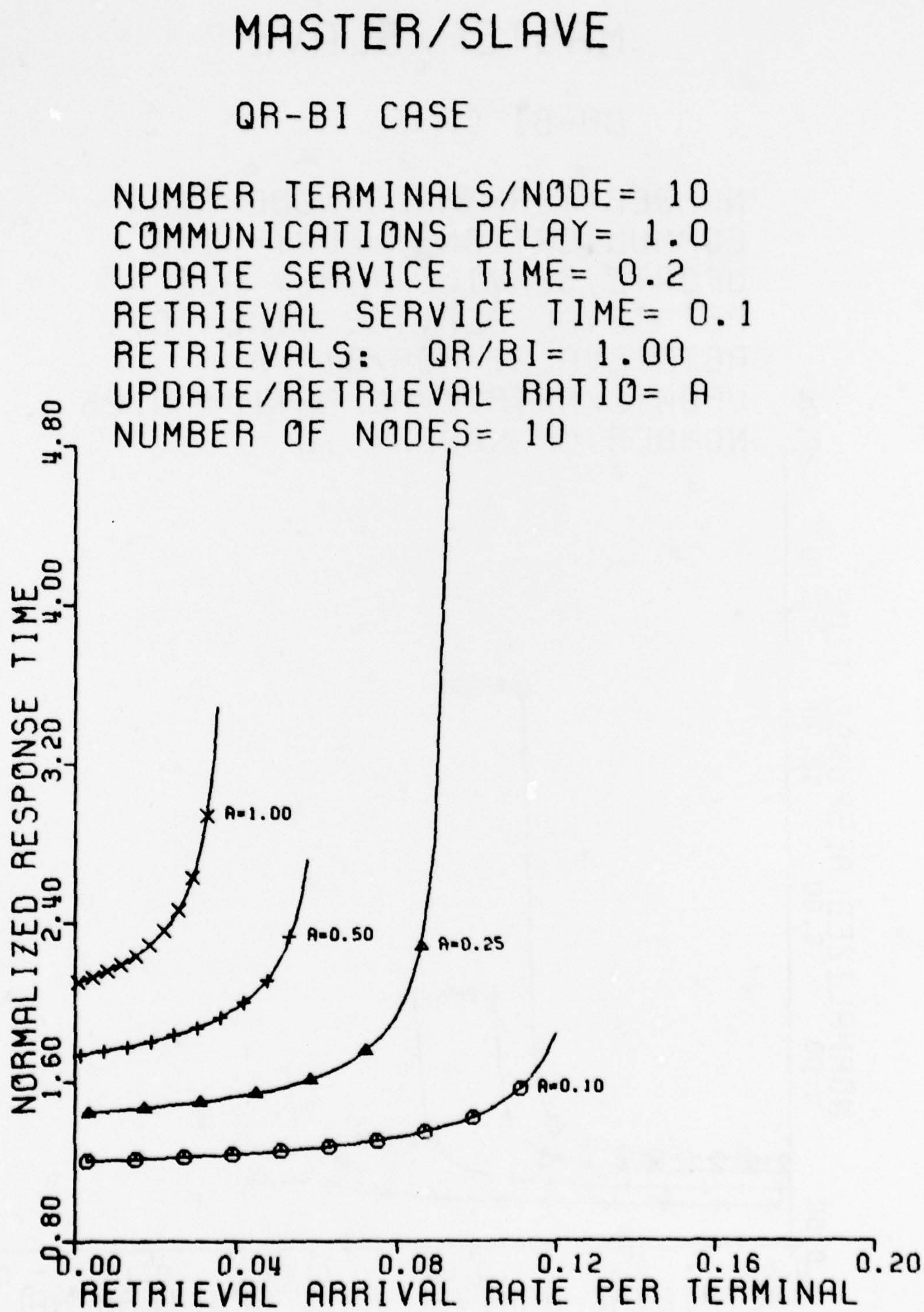


Figure 4-17. Response Time Dependence on the Update/Retrieval Ratio



4.3.3 Synchronized Management

LeLann has suggested [LELA78] that synchronized management can be handled by explicitly sequencing update requests using tickets similar to ones found in a bakery to assign service order to the customers. The ticket order ensures that updates are processed in exactly the same order throughout the distributed system. To eliminate concurrency in accessing (as well as to avoid deadlock over data resources) and still have distributed control over the distributed database, the database hosts of the system are arranged in a virtual ring by assigning permanent identification numbers in a sequential increasing fashion around the ring so that a predecessor and successor are defined for each node. A special message called the control token is circulated around the virtual ring to implement the ticketing facility. Tickets can be taken from the "dispenser" (the control token) only by the node owning the token and are assigned to pending update requests only after the control token has been accepted by that node's ring successor. Thus when an update request arrives at a particular node, it is assigned the next ticket that the node has available. If there are no tickets left, the request must wait in a pending queue until the control token arrives (back at that node) and the node can get more tickets.

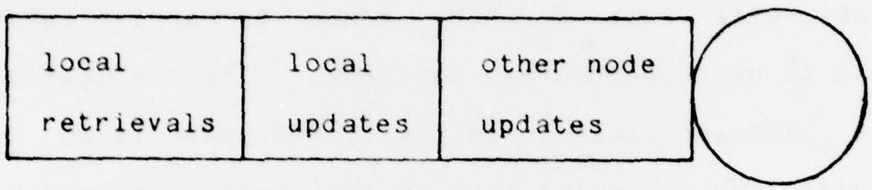
Ticketed update requests are sent throughout the system. All nodes are restricted to applying updates in ticket order, so if preceding ticket numbers are missing, a newly arriving ticketed update must be queued until all lower ticket numbers have been processed. The token effectively acts in place of a centralized, master clock to provide global event sequencing.

Synchronized management schemes have not been simple to model [GARC78]. Even the logically simple virtual ring does not model directly into a simple queuing system. An arriving update transaction not only waits for prior transactions queued for service, it may also wait to get a ticket assigned, it must wait for prior ticketed updates to be applied, and it may have to wait to see if prior ticket numbers are used or have expired. We will avoid some of this complexity by choosing to work with a very simple logical representation of the virtual ring, a sequential ring with the successors located in simple ring order. Even in fully interconnected networks (where each node has a direct communication link to each other), we might want to assign successors according to physical proximity. The sequential ring allows us to continue using the assumption of constant, fixed communications delays without consideration of routing and network interconnection patterns. We will see that the solution is efficient enough to compete with the other management schemes only under certain conditions. For now our concern is that it is simple enough to be modeled.

Update and retrieval transactions will be considered as Poisson arrivals uniformly distributed over all network terminals, as before. If we do not allow tickets to be requested ahead in anticipation of future arrivals, the update requests will wait for the token to arrive in order for tickets to be requested and assigned. The simplest scheme to ensure update application in the ticket order is to circulate the updates by appending them to the token and removing them when they get back to their originating node. This means that updates arrive at each node all at once to be processed in a batch.

Retrieval requests will still be handled locally at each node (the complete duplicate copies assumption). In order to be sure the retrievals get up-to-date information they will have to wait for the current circulating set of updates to be applied. This means retrieval requests will also be saved until the token arrives and then be added to the batch behind the current set of updates, Figure 4-18. The flow diagram of Figure 4-19 shows how all the nodes are connected, while the internal details for a single node are in Figure 4-20. Notice that even this simplified node model is not a standard queuing system, since service may not begin until the token arrives and transactions arriving after batch processing begins may not join the queue being processed. This aspect is emphasized in Figure 4-20 by the barrier which is triggered for transmission only by the arrival of the control token (think

Figure 4-18. Queue Model for Basic Synchronized System



of the diagram as a hybrid of flow control and an evaluation net). In order to have a stable system that doesn't bog down, we are also constrained to complete processing of one batch before the next arrives.

The average response time for a transaction will be made up of three parts:

* waiting for the token,
* waiting for the predecessors in the queue to be served, and
* actual service time.

Using our assumption of activity being uniformly distributed across all terminals and nodes of the network, we will say that the average wait for the token is half of the time for a virtual circuit. The order in the queue is updates from all other nodes, local updates and then local retrievals (Figure 4-18), so all local transactions wait for non-local updates (from the other N-1 nodes) to be processed. Since average

Figure 4-19. Network Flow Diagram for Synchronized Management

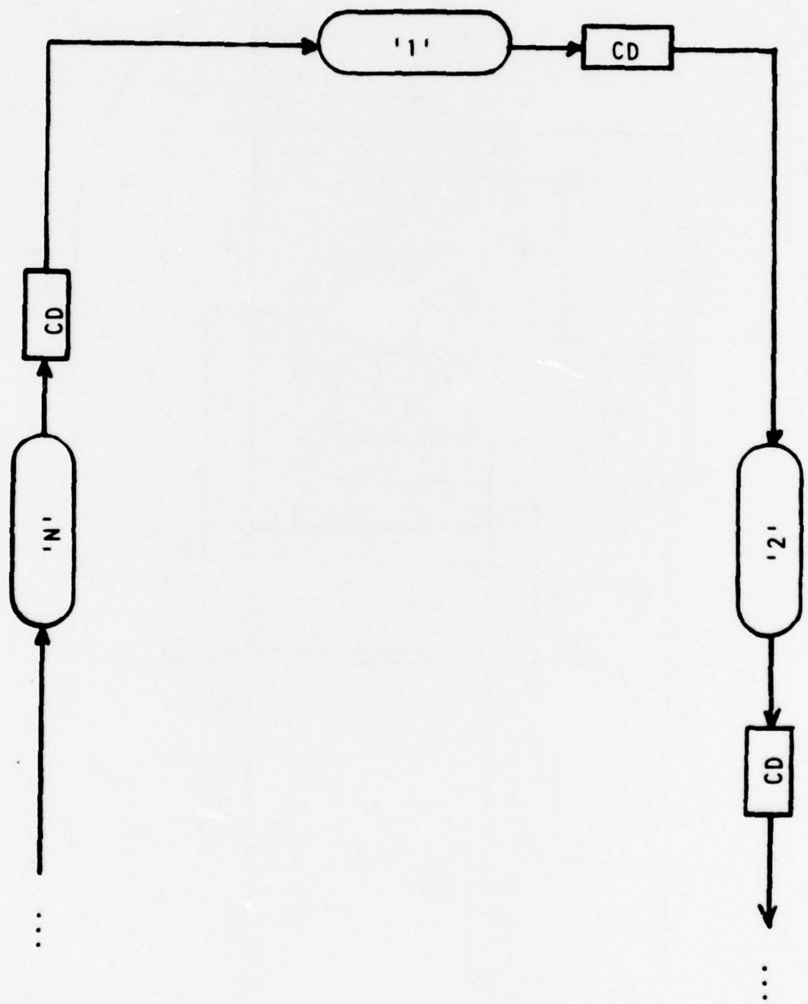
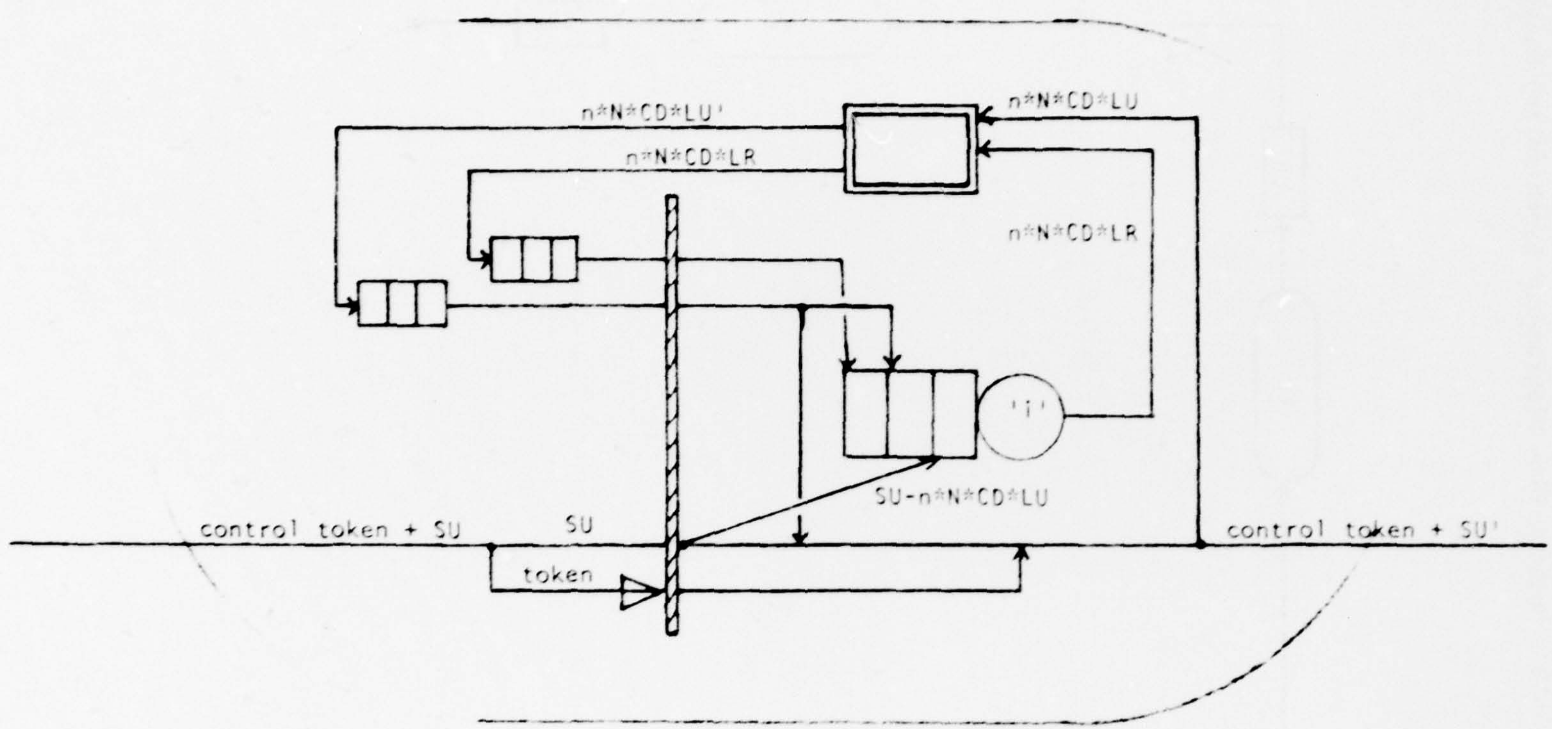


Figure 4-20. Internal Node Flow for a Basic Synchronized System



SU = N * n*N*CD*LU

SU'= SU - n*N*CD*LU + n*N*CD*LU'

service times are constant, the uniform distribution says the average wait for local updates will be for half the total number and the average wait for local retrievals will be for all updates plus half the number of retrievals. Thus we end up with (for details, see section B.1.3 of appendices)

RT = .5*N*CD + n*(N-1)*N*CD*A*LR*XU
+ [A/(A+1)] * (.5*n*N*CD*A*LR*XU)
+ [1/(A+1)] * [n*N*CD*LR*(A*XU+.5*XR)] .

Figure 4-21 shows how this response time varies with the number of nodes and Figure 4-22 shows the dependence on the update/retrieval ratio.

If we treat the control token as having the updates to be circulated appended to it, the combined token/update package can be considered as a message. For each update, then, N messages are required to complete the circulation, and the number of messages averaged over both updates and retrievals gives us

NM = [A/(A+1)] * N .

Figure 4-21. Dependence of Response Time on the Number of Nodes in the System

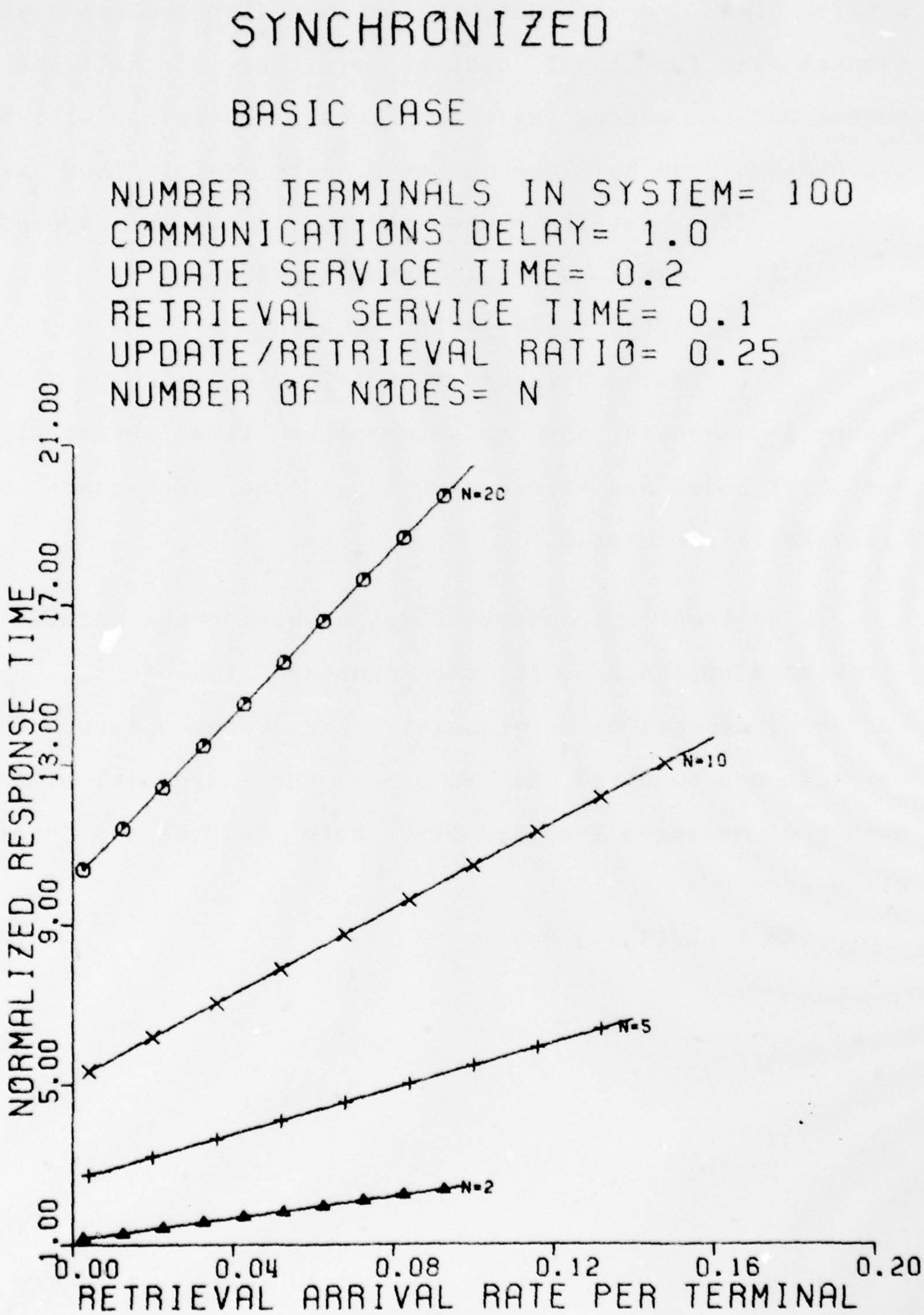
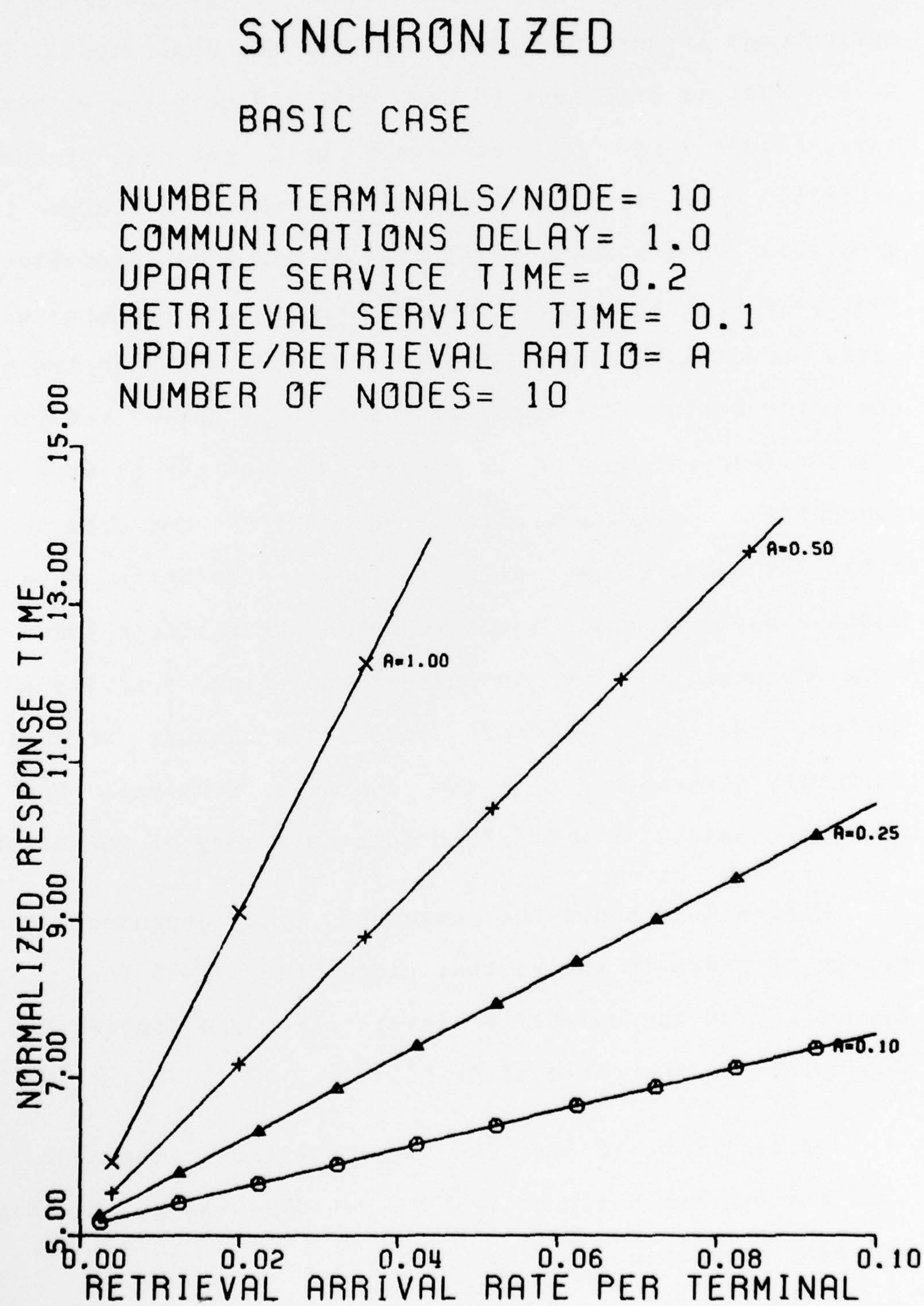


Figure 4-22. Dependence of Response Time on the Update/Retrieval Ratio



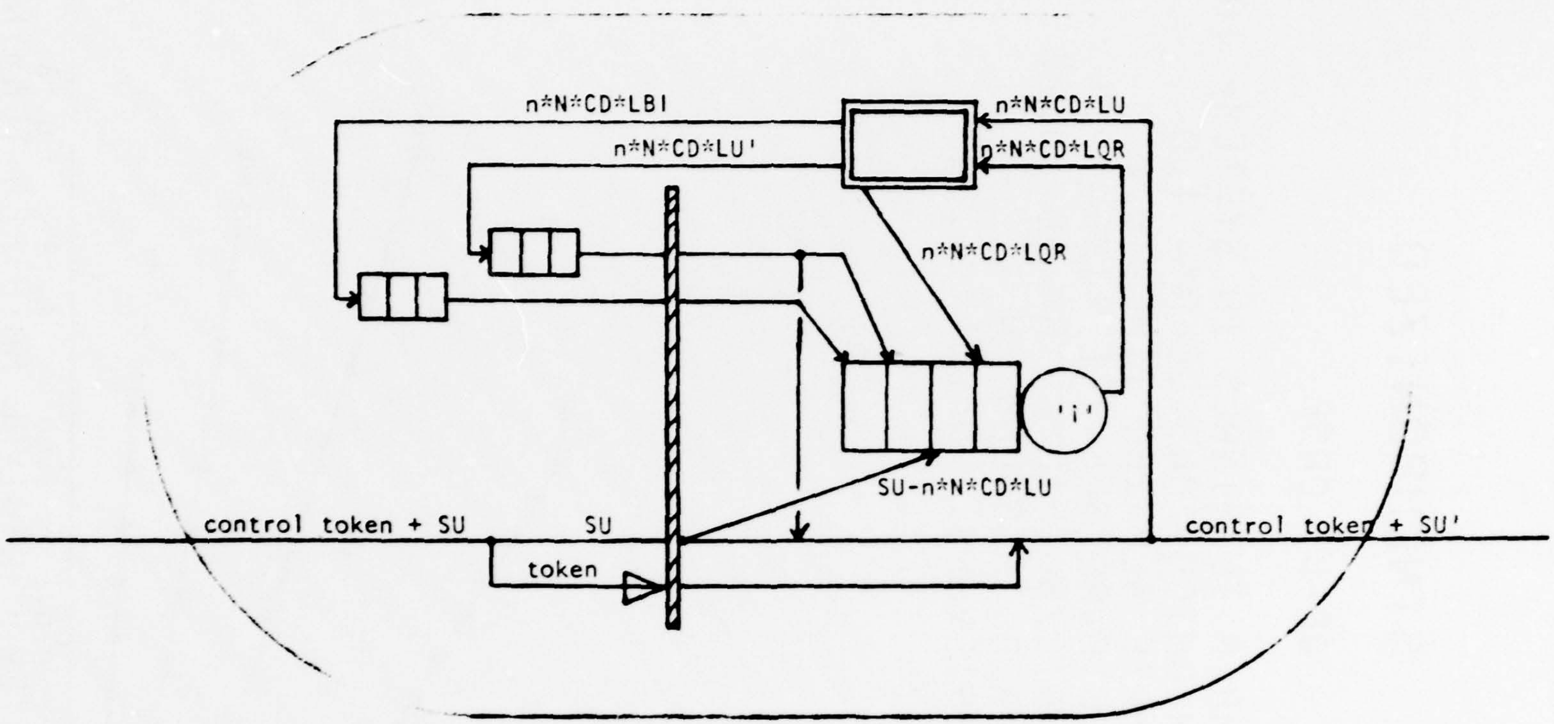
The network flow for synchronized management in a QR-BI environment is just the same as for the basic model, Figure 4-19. What is different is the queuing discipline within each node, Figure 4-23. QR retrievals will get head-of-the-line priority treatment if they arrive while a batch is in progress. Otherwise they will simply queue by themselves for immediate processing service. BI retrieval requests will be saved up until the token arrives and then added to the end of the batch behind the updates, so that their results will provide information that is "best" within the context of the currently circulating update batch. Any BI requests arriving after the token (even if it's during the batch processing) must be saved up until the next token arrival to ensure "best" with respect to the concurrent updating activity in the system. As in the basic model, we assume arrivals are uniformly distributed over the system's terminals and that each node maintains a complete duplicate copy of the database.

Figure 4-24 shows the response time dependence on the number of nodes in the virtual ring, Figure 4-25 the dependence on the update/retrieval ratio, and Figure 4-26 the dependence on the ratio of QR/BI.

As in the basic case for synchronized management, it is only the updates which contribute network message traffic to the cost prediction,

$$NM = [\ A/(A+1)\ ] * N .$$

Figure 4-23. Internal Flow Diagram for Synchronized QR-BI Node



SU = N * n*N*CD*LU

SU'= SU - n*N*CD*LU + n*N*CD*LU'

Figure 4-24. Response Time Dependence on the Number of Nodes

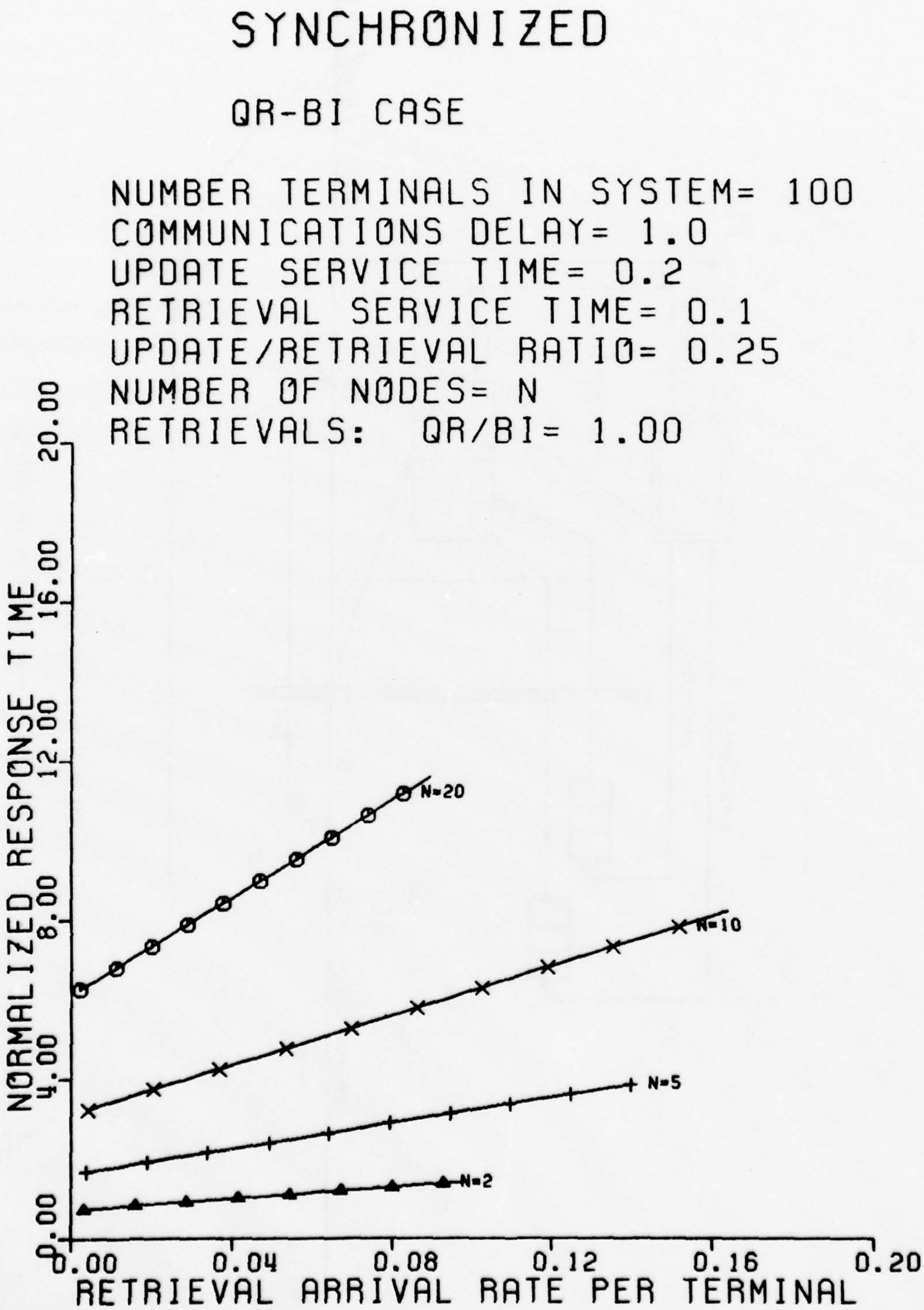


Figure 4-25. Dependence of Response Time on the Update/Retrieval Ratio

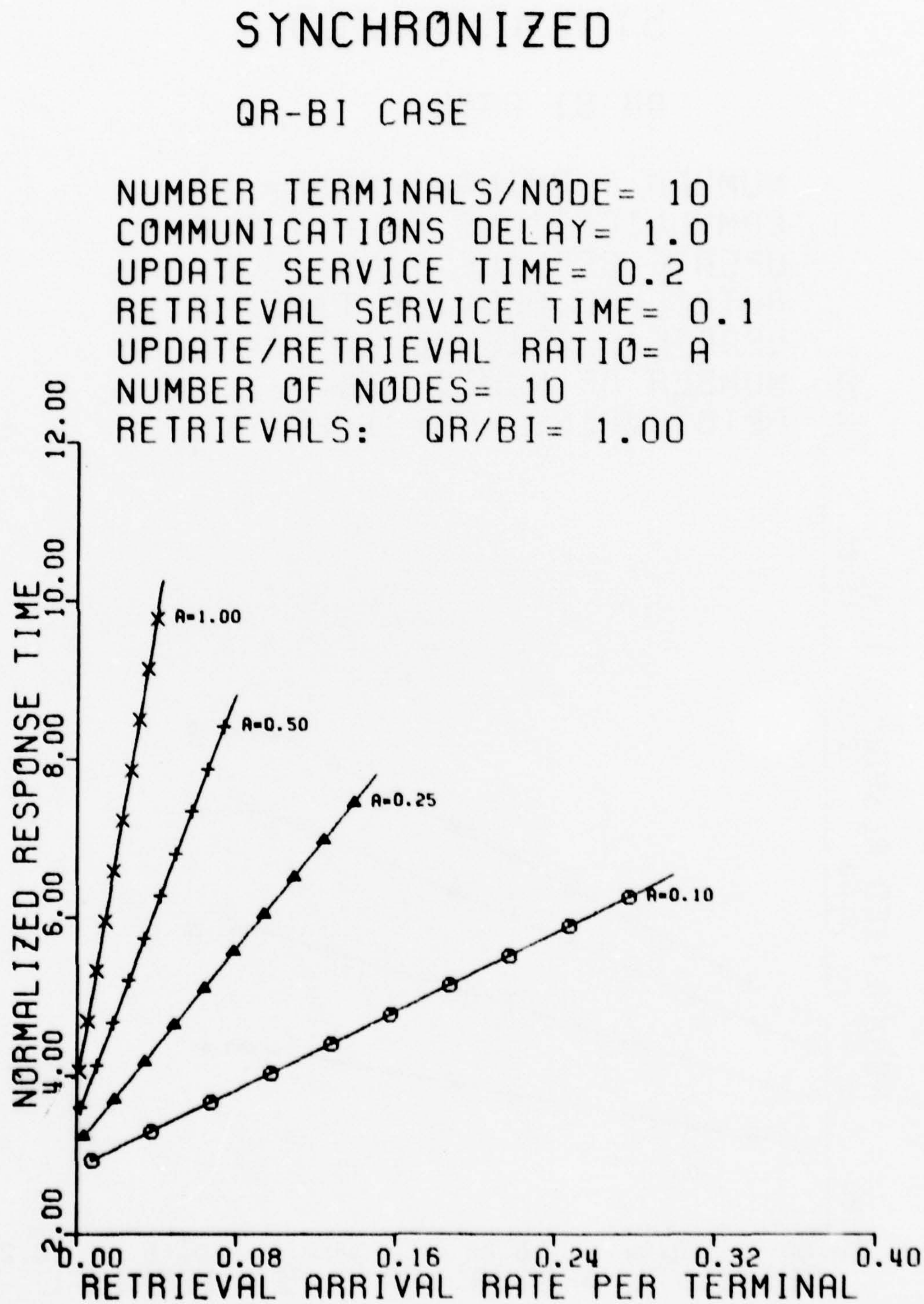
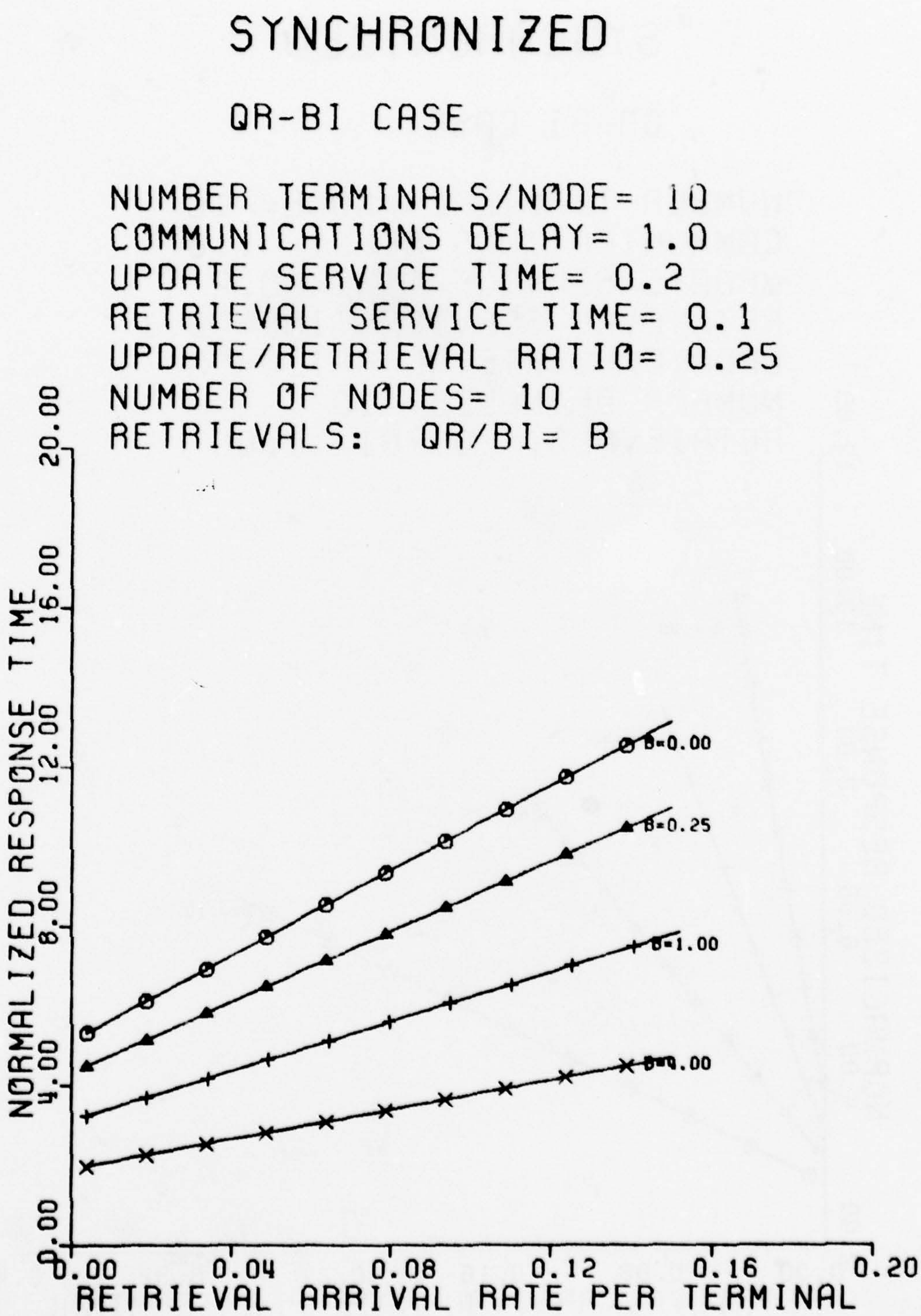


Figure 4-26. Response Time Dependence on the QR-BI Ratio



4.3.4 Delayed Synchronization Management

There are really three phases for us to be concerned with in delsync management:

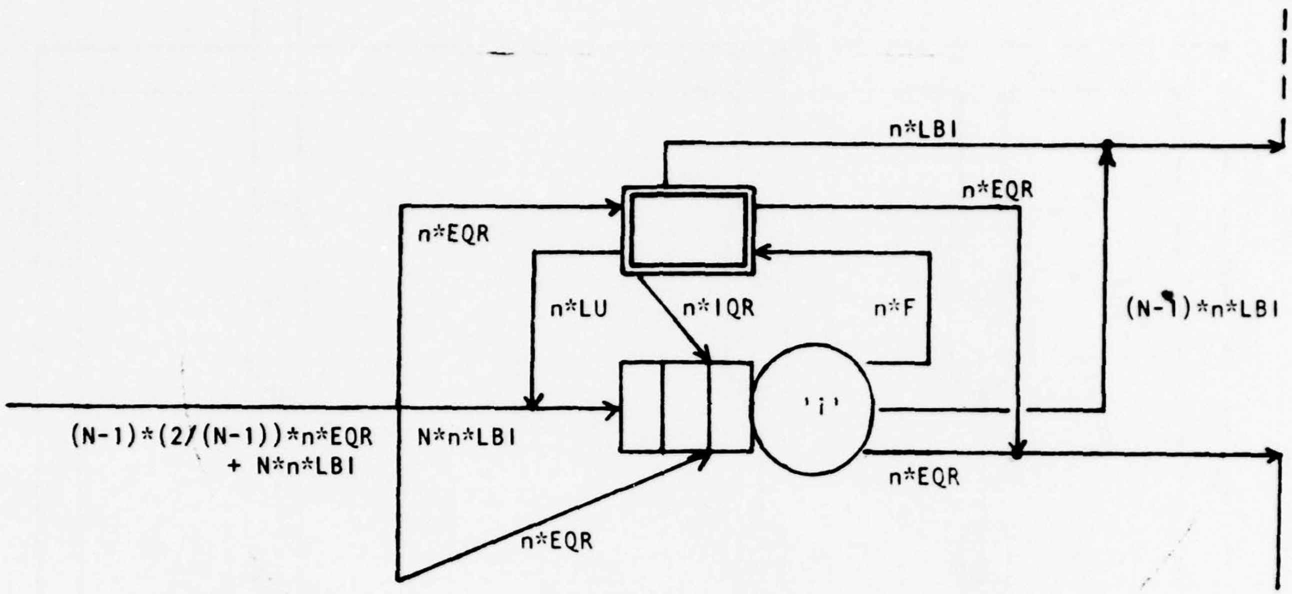
* ordinary daily operation: where updates are all processed locally, most QRs are processed locally (because the copies diverge doing local updates some QRs may need data not stored in the local copy), and BI results must be collected from all other nodes so that the most recent result can be selected;
* merging: the periodic technique by which all the divergent copies are synchronized to the most recent information across the system; and
* transition: which is actually the end of a file's delsync classification as it changes to synchronized.

The phase which is important to the cost/performance predictions we have been making in this chapter is the daily operation phase; in fact, we will assume that merges are done during some period of very low activity such as the middle of the night. For applications such as world-wide airline reservations, there may not be any such period, and we really should account for some extra processing contention and delay when merges take place.

The queuing model for each delsync node is shown in Figure 4-27. We see that the arrival rate of QR requests for which the required data are local is IQR = LQR * p(local) and these get high priority treatment in the local queue. Actually, the probability of finding data locally is a function of time: right after a merge, p(local) = 1; just before a merge, p(local) has its minimum value, less than 1. For our first-order statistics, we will use only an average value, p(local), and apply it on the average over the entire time period between merges. Thus, the rate at which QR requests must be passed to another node is EQR = LQR * [1-p(local)]. We assume these external requests will be uniformly distributed over the remaining nodes. This is shown by the solid lines of Figure 4-28 along with the communications delays incurred. Similarly the local node's share of QR requests processed for the other nodes is returned directly to those nodes via the solid lines. This is slightly different from the control flow algorithm of section 3.3.3; it is equivalent under the uniform fixed communication delay that we have assumed between any two nodes in the network. That is, we are now adding an assumption that the identity of the copy responding "quickest" will be uniformly distributed over the remaining nodes.

The dashed lines of Figure 4-28 represent virtual ring connections which are used to handle BI retrievals. Local BI requests are put out onto the virtual ring, as are BI results

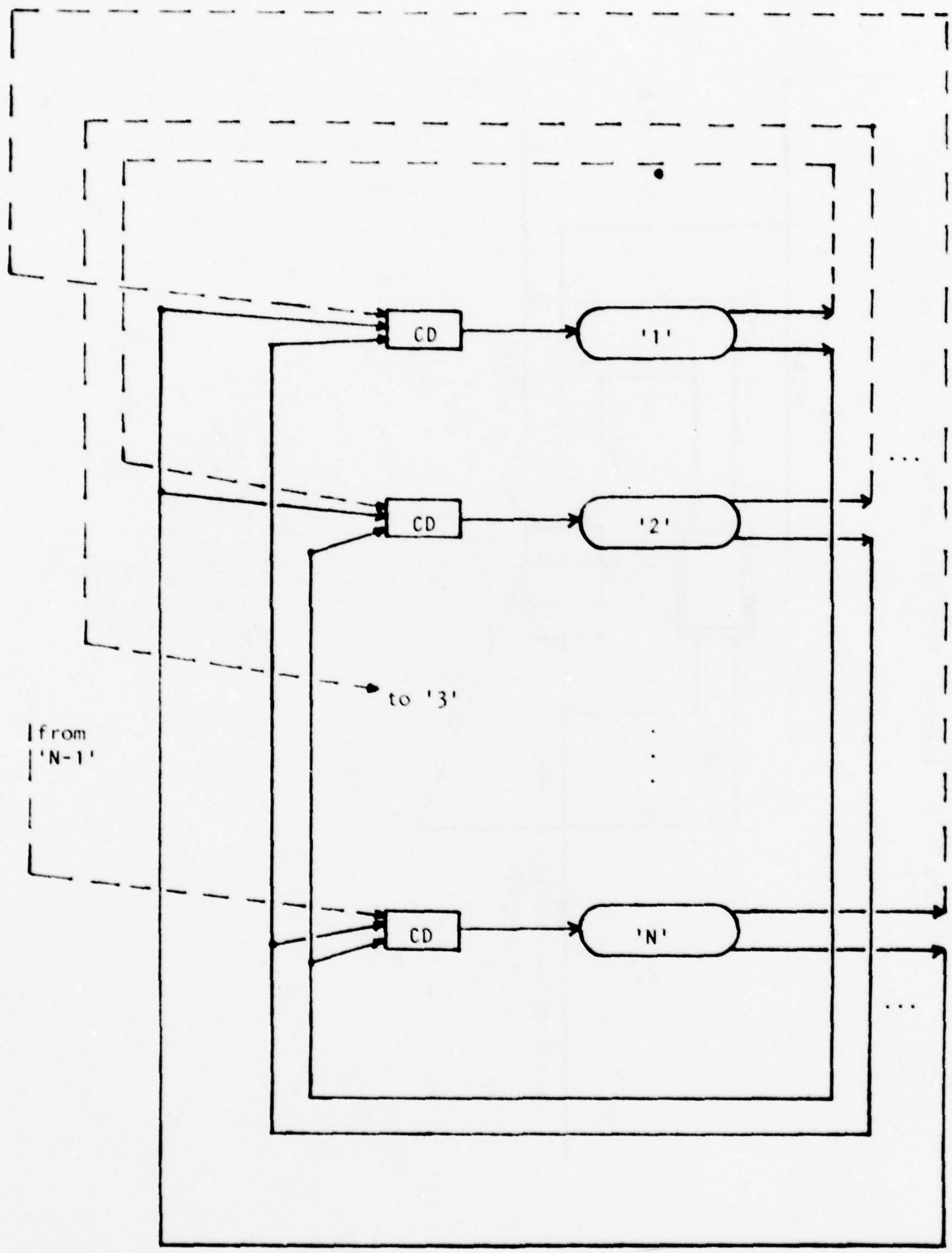
Figure 4-27. Internal Flow Diagram for Delsync System



IQR = LQR*P(local)
EQR = LQR*(1-P(local))
F = LU + IQR + LBI

$1 \leq i \leq N$

Figure 4-28. Overall Flow Diagram for Delsync System



going back to the other nodes (Figure 4-27). Results from the ring of previous local requests sent out are processed locally to choose the best result and are then returned to the user. Again this differs from the control flow algorithm of section 3.3.3 for modeling tractability.

We can express the average delsync response time (developed in detail in section B.2.4 of the appendices) as

$$RT = TU * [A/(A+1)] + N * (CD + TBI) / [(A+1)*(B+1)] + \{TQR+[1-p(local)]*2*CD\} * B/[(A+1)*(B+1)] .$$

Figure 4-29 shows how the response time varies with the probability that QR retrievals can be handled locally. Figure 4-30 shows the tradeoff between more nodes handling more arrivals with higher average response times because of the longer time to complete a circuit of the virtual ring. Dependence of response time on the proportion of QR to BI retrieval requests is dramatically demonstrated in Figure 4-31: the more activity that can be handled locally, the better the response. Figure 4-32 shows the dependence on the update/retrieval ratio.

Network messages are required to handle both the best information retrievals and the small portion of quick response retrievals which cannot be answered from locally stored information. The average cost is thus

$$NM = \frac{A}{A+1} * N + \frac{B}{(A+1)*(B+1)} * [1-p(local)] * 2$$

Figure 4-29. Dependence of Response Time on Locality

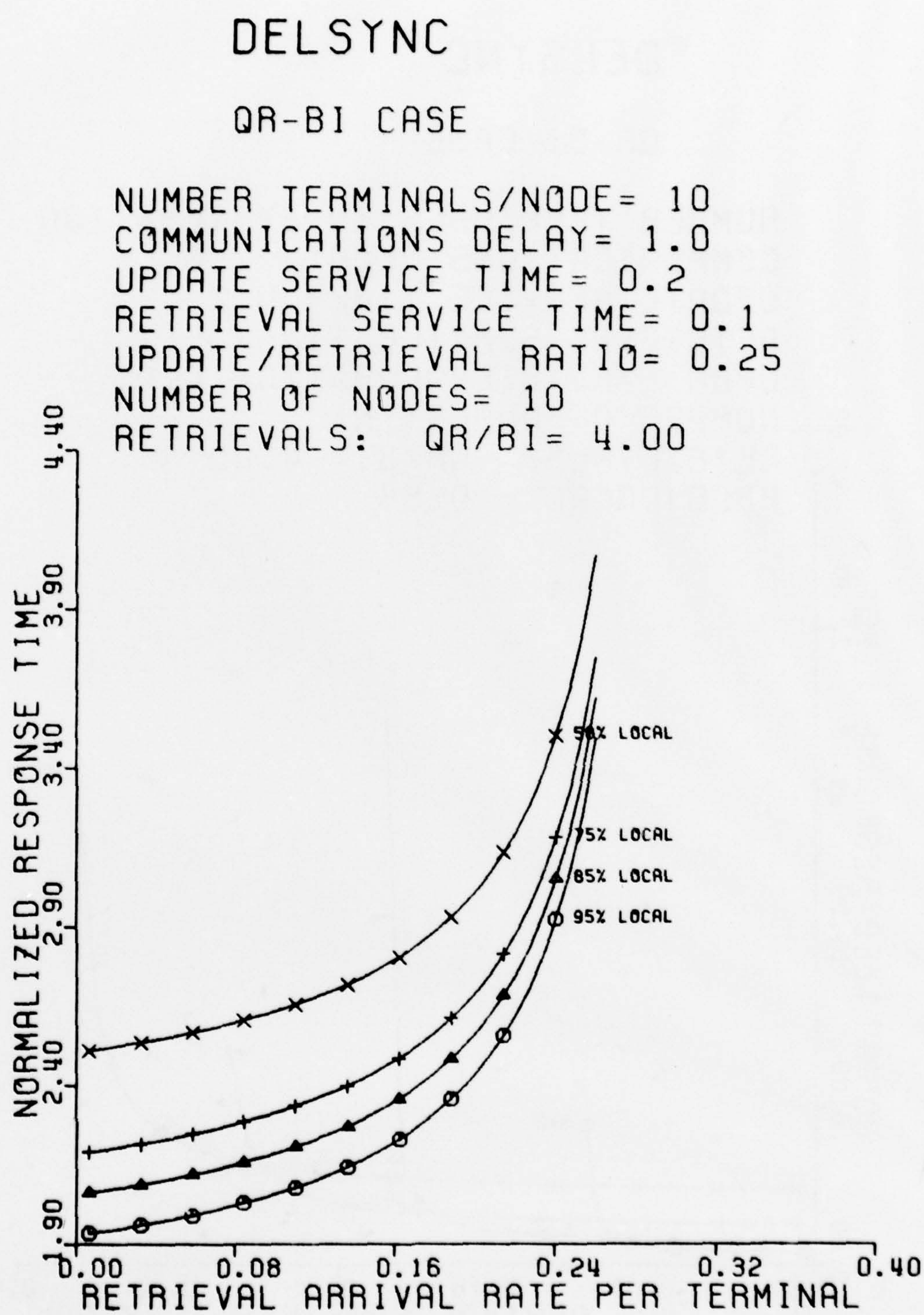


Figure 4-30. Dependence of Response Time on the Number of Nodes in the System

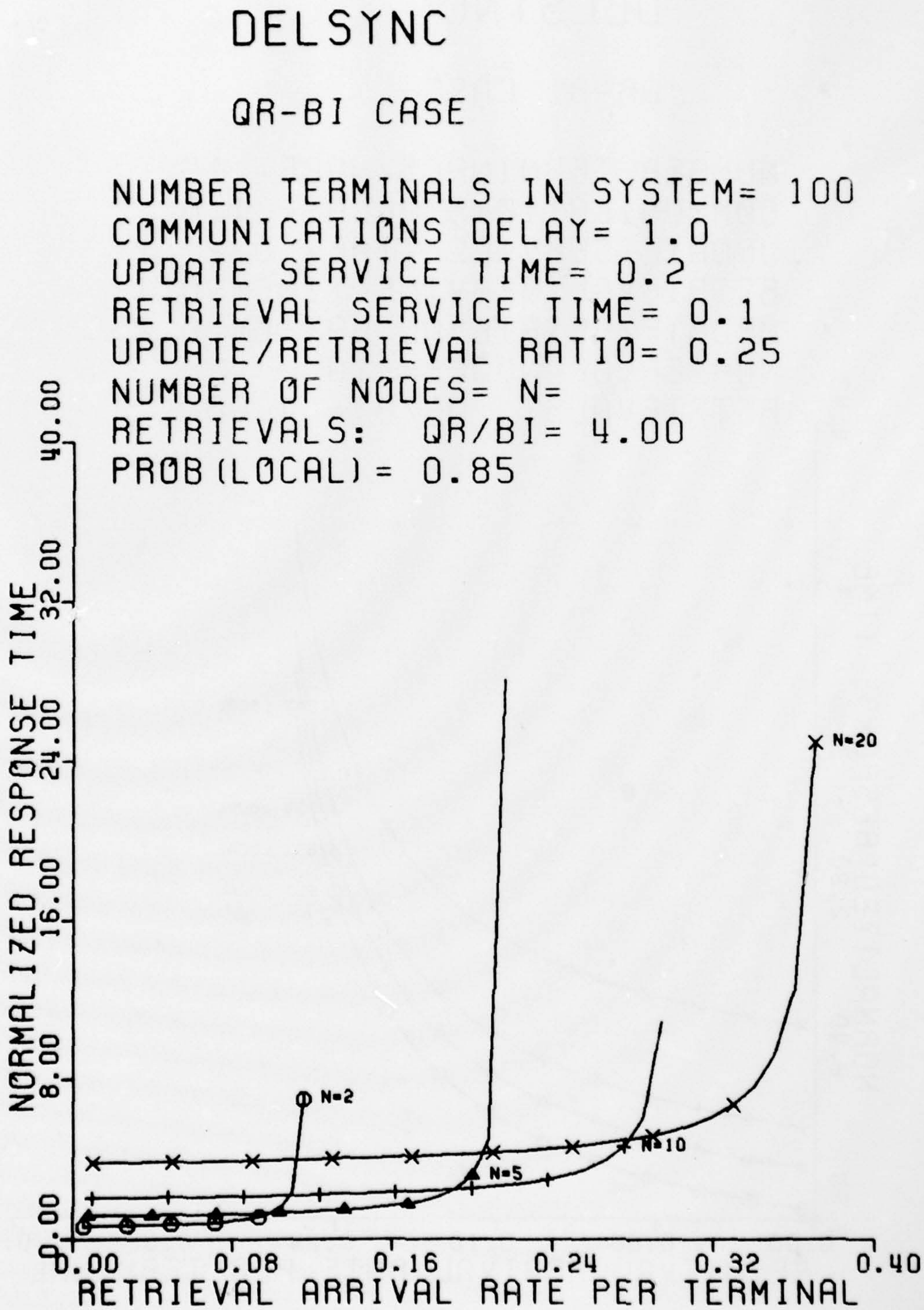


Figure 4-31. Dependence of Response Time on the QR/BI Ratio

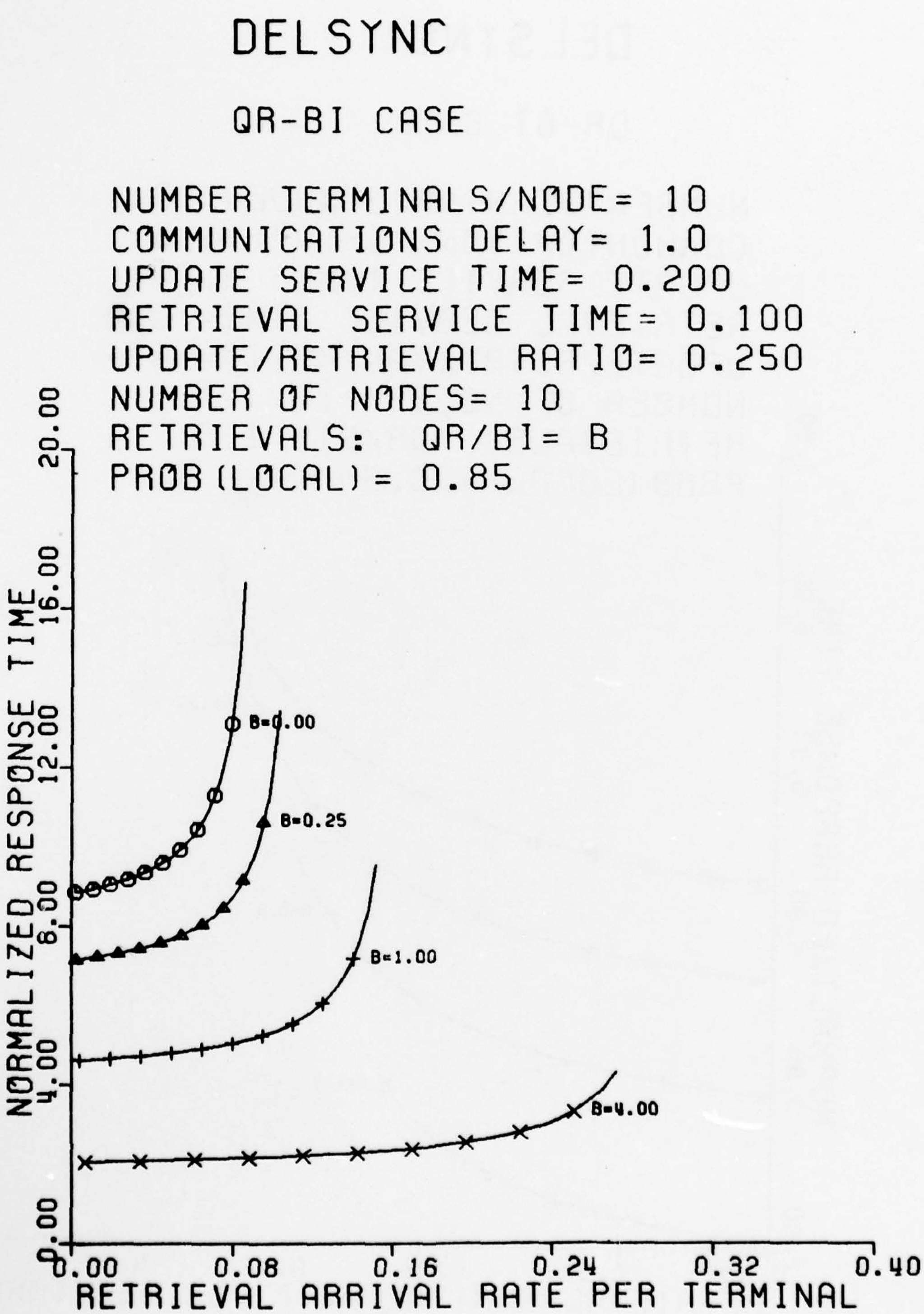
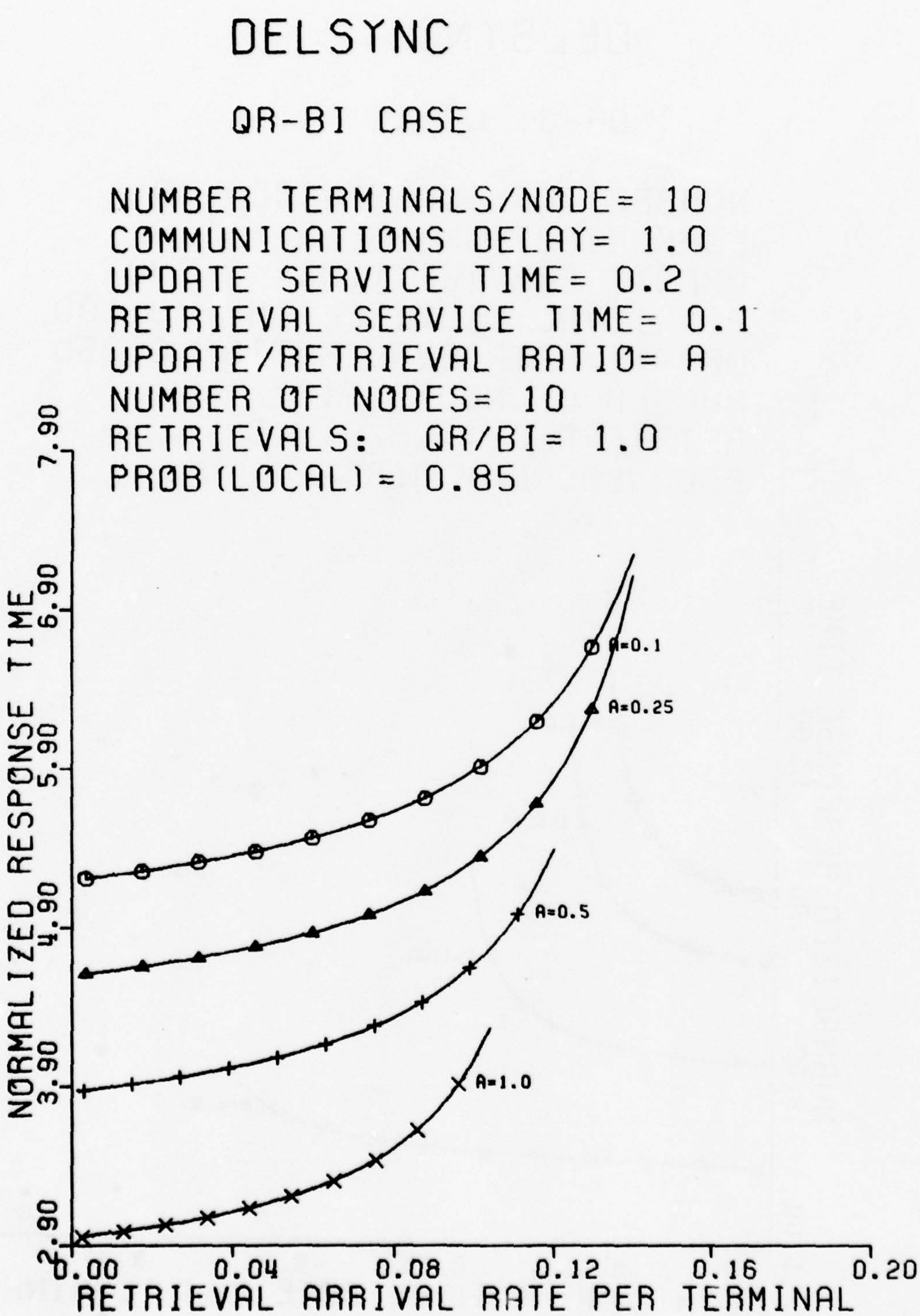


Figure 4-32. Response Time Dependence on the Update/Retrieval Ratio



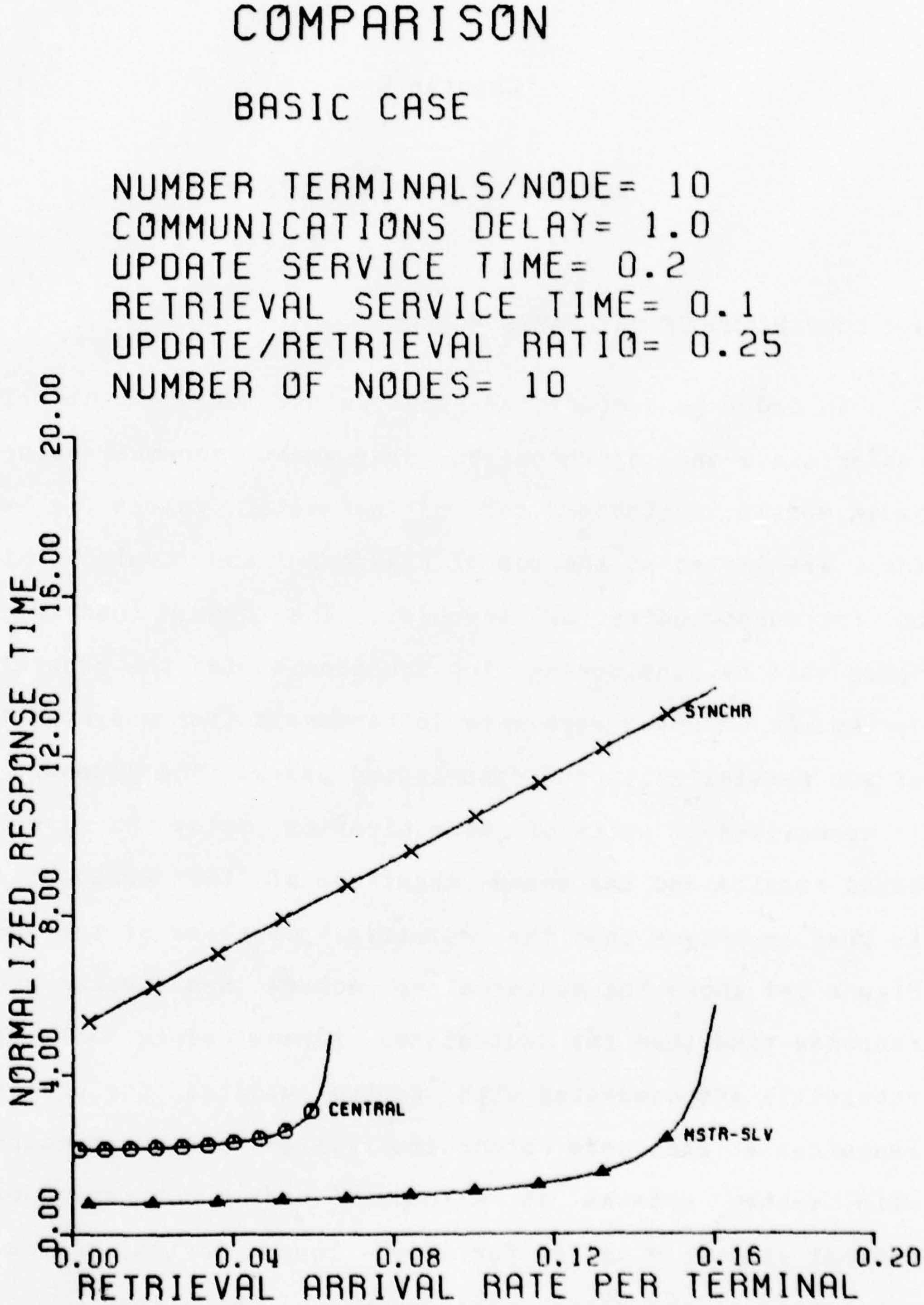
Chapter 5

RESULTS OF THE ANALYSIS

5.1 COMPARISON OF THE BASIC MODELS

In order to compare the performance of the centralized, master/slave and synchronized management schemes using the basic models, a standard set of parameter values is used. These are listed at the top of Figure 5-1 and could be taken to represent units of seconds. The system load is kept comparable by considering 100 terminals in the centralized system and 10 nodes each with 10 terminals (for a system total of 100 terminals) in the distributed cases. The response time is normalized to units of communication delay to give lower bound results and the actual magnitude of the response times is less important than the relative positions of the curves. Figure 5-1 shows the master/slave scheme has better average response time than the centralized scheme since only local retrievals are competing with system updates for processing resources at each node rather than system retrievals competing with system updates at a single node. Saturation of the central site also occurs for much lower arrival rates than saturation of the master node because of the higher

Figure 5-1. Comparison of the Basic Models



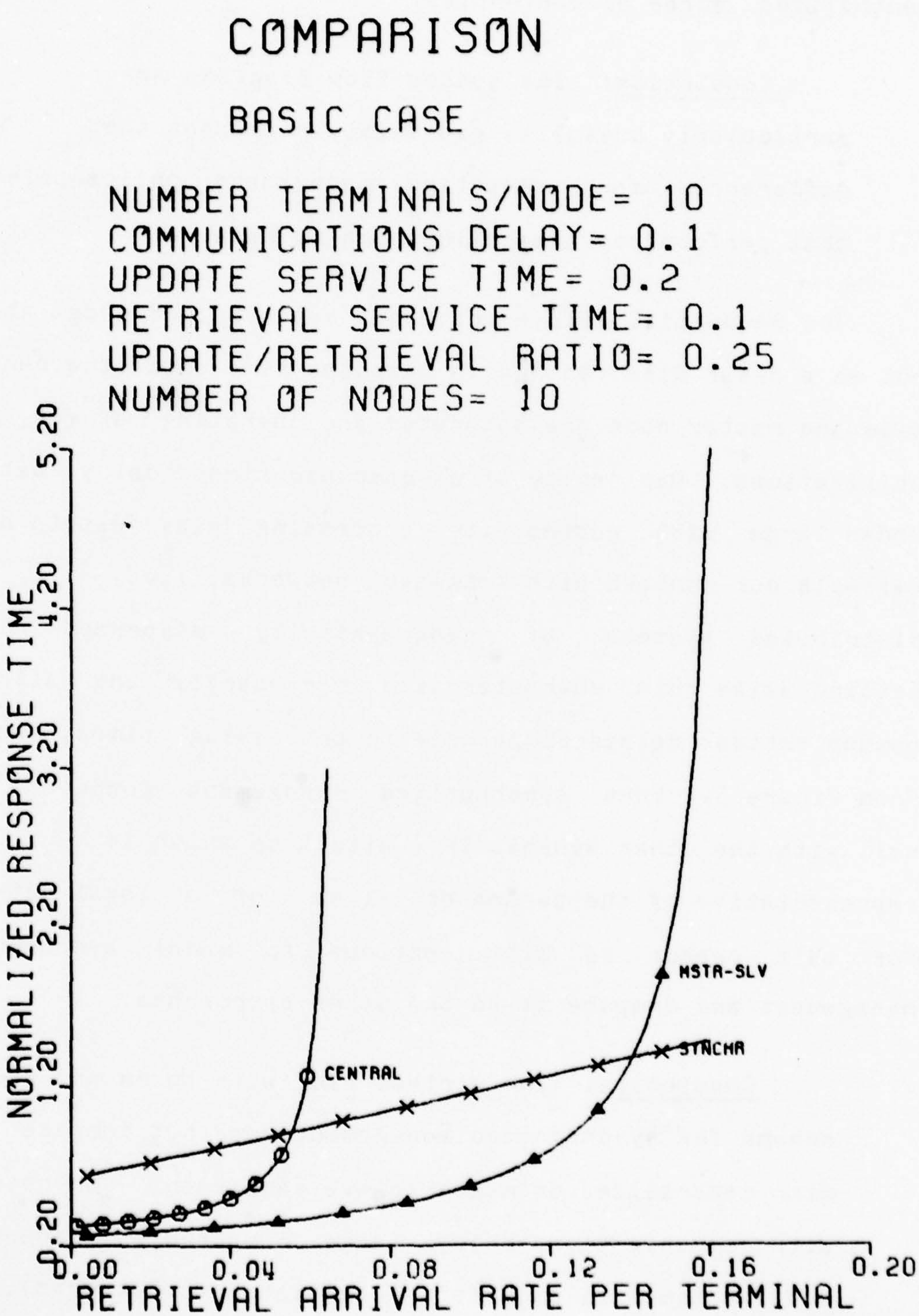
centralized degree of contention.

<u>Conclusion</u>: The system flow diagrams are particularly useful to explicitly represent the differences among competing management philosophies so that performance predictions can be made.

The sequential nature of the synchronized model stands out as a clear disadvantage in response time until the central site and master node are saturated and operating at very high utilizations. Our choice of a communications delay between nodes large with respect to processing delays within nodes reflects our concern with long-haul networks, i.e., distributed systems of geographically dispersed nodes. Setting aside this characteristic momentarily and allowing communications delays comparable to processing times, we see from Figure 5-2 that synchronized management competes very well with the other schemes in a situation which is representative of the parameter values for a local network. For this reason we will continue to model synchronized management and compare it to the other approaches.

<u>Conclusion</u>: The virtual ring with token and tickets scheme for synchronized management does not compare well with centralized or master/slave management for networks with communications delays large compared to transaction service times (i.e., for long-distance networks). It does compare well for the smaller communications delays

Figure 5-2. Another Comparison of the Basic Models

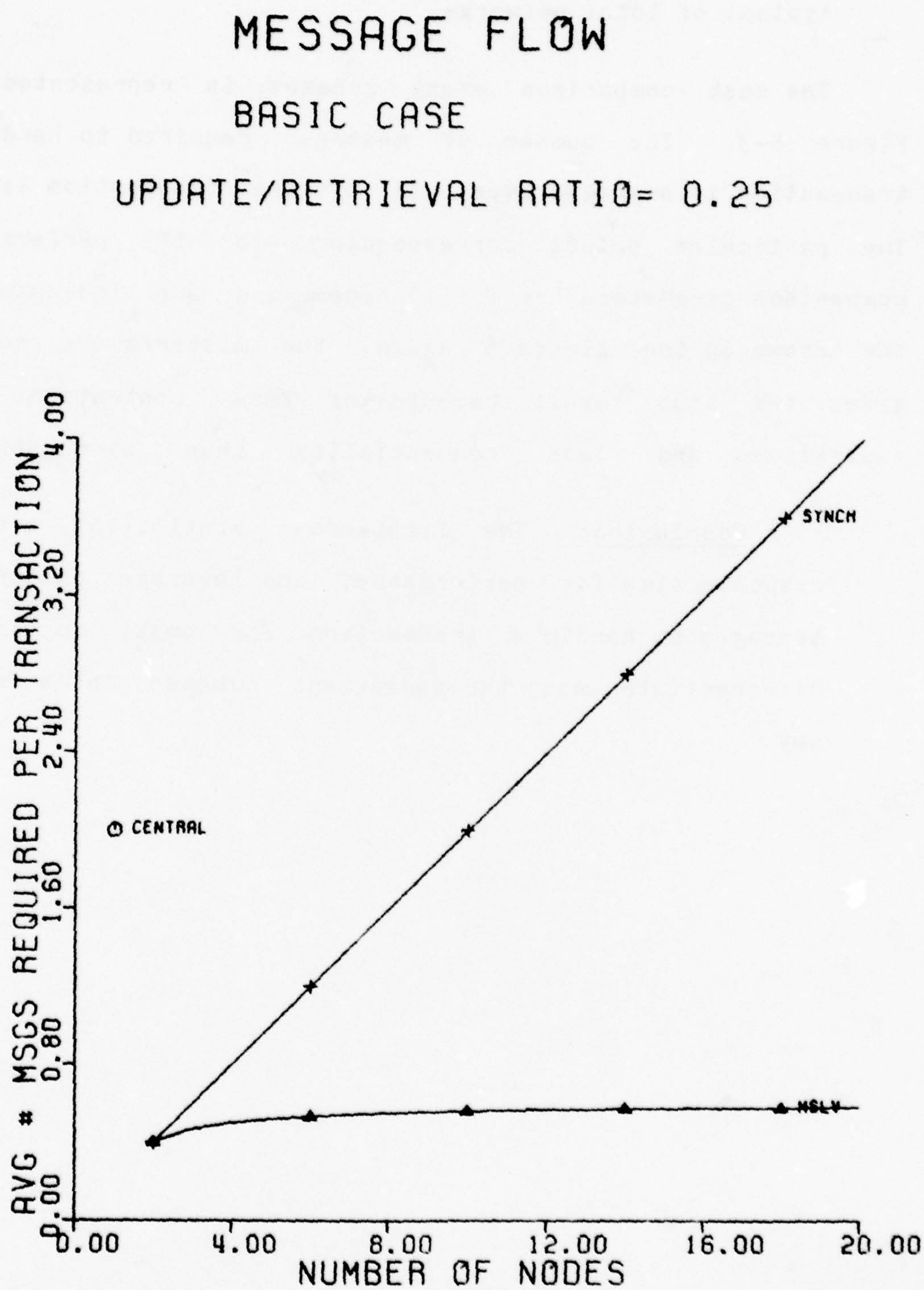


typical of local networks.

The cost comparison among schemes is represented in Figure 5-3. The number of messages required to handle a transaction is averaged over all of the transaction types. The particular points corresponding to the performance comparison parameters are for 10 nodes and are indicated by the arrows in the figure. Again, the master/slave scheme gives the best result because of less contention than centralized and less sequentiality than synchronized.

<u>Conclusion</u>: The first-order statistics, average response time for performance, and average number of messages to handle a transaction for cost, do indeed differentiate among the management schemes in a formal way.

Figure 5-3. Message Flow Comparison for Basic Models



5.2 COMPARISON BETWEEN BASIC AND QR-BI MODELS

A direct comparison for our standard parameter set of the basic and QR-BI models of master/slave management is shown in Figure 5-4. It clearly shows that introduction of BI retrievals, which must be serviced by the master instead of locally by the slaves, introduces additional resource contention in the master. This degrades the average system response time, and increasingly so as the QR/BI proportion decreases. The average number of messages required to process a transaction, Figure 5-5, reflects the same effect. The point is that the system performance is completely driven by the performance of the master node, because it has the most work to do.

In contrast, the average response time for synchronized management, Figure 5-6, shows significant improvement with increasing amounts of QR processing that can be done between batches of BIs and updates. Since only updates require messages for circulation, the average number of messages per transaction is identical for both the basic and QR-BI models.

Delayed synchronization has no basic model since it was designed specifically for the QR-BI case. We can refer back to Figure 4-31 to see how it improves with an increasing proportion of QR.

Figure 5-4. Performance Comparison of Master/Slave Models

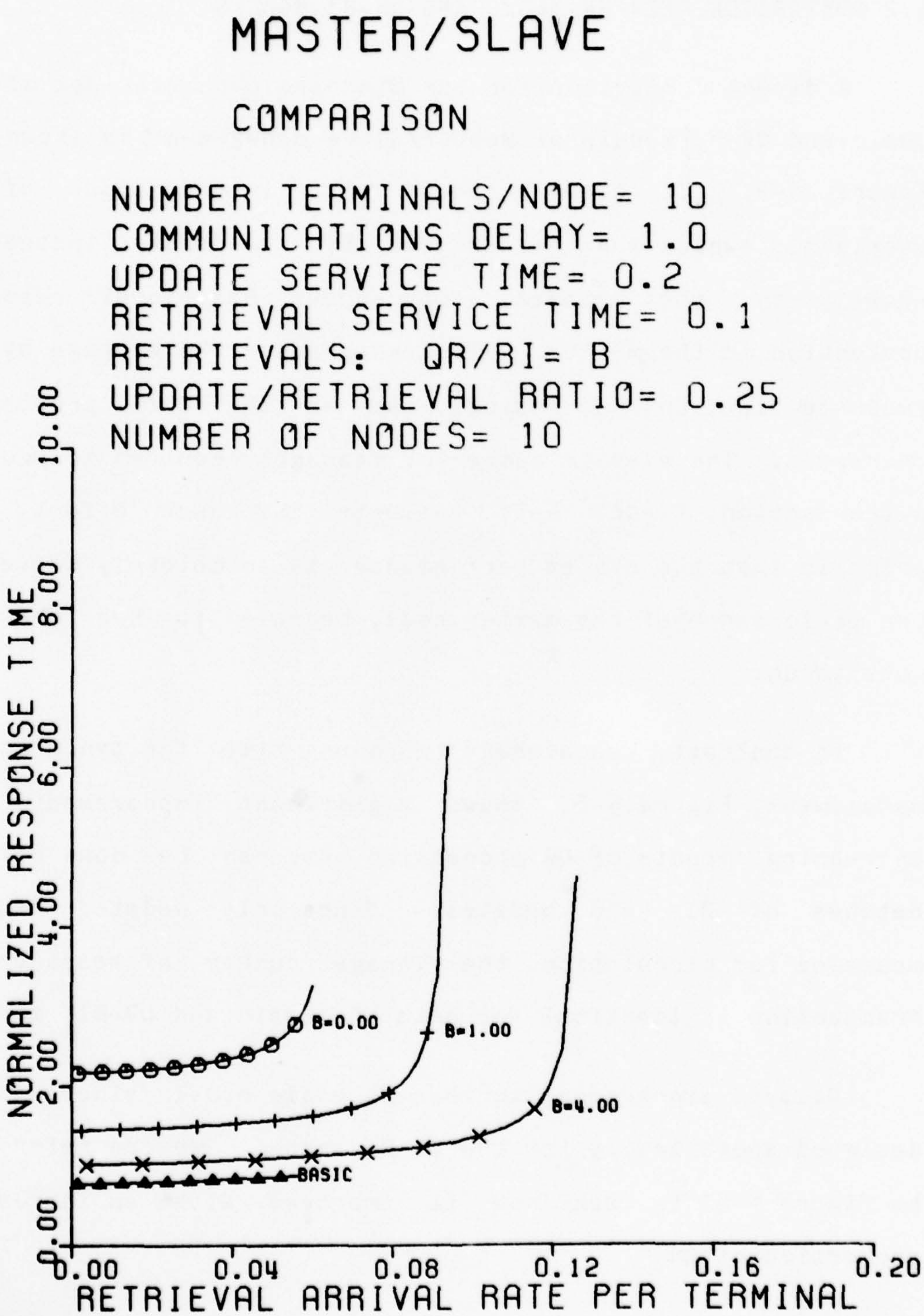


Figure 5-5. Cost Comparison of Master/Slave Models

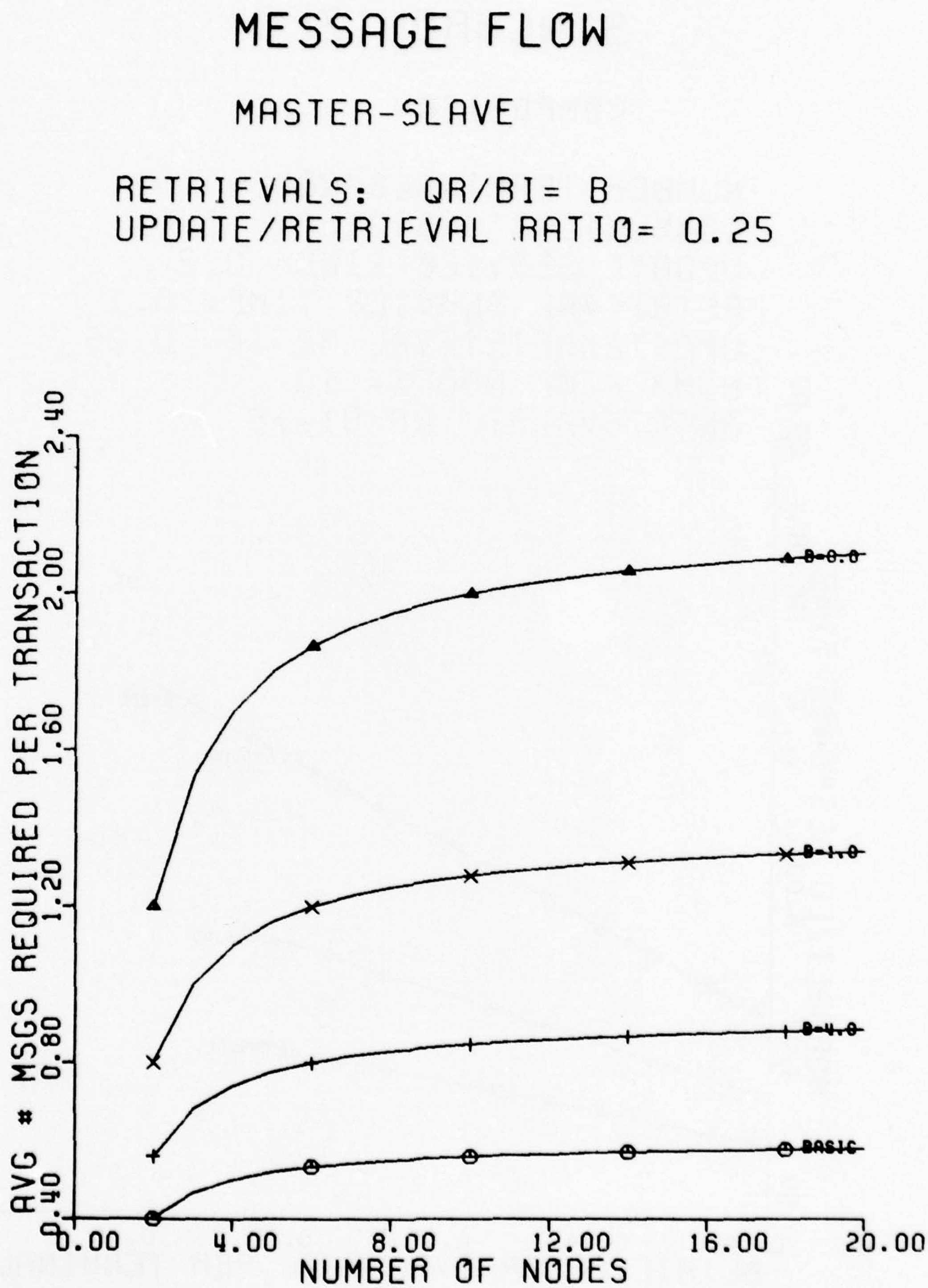
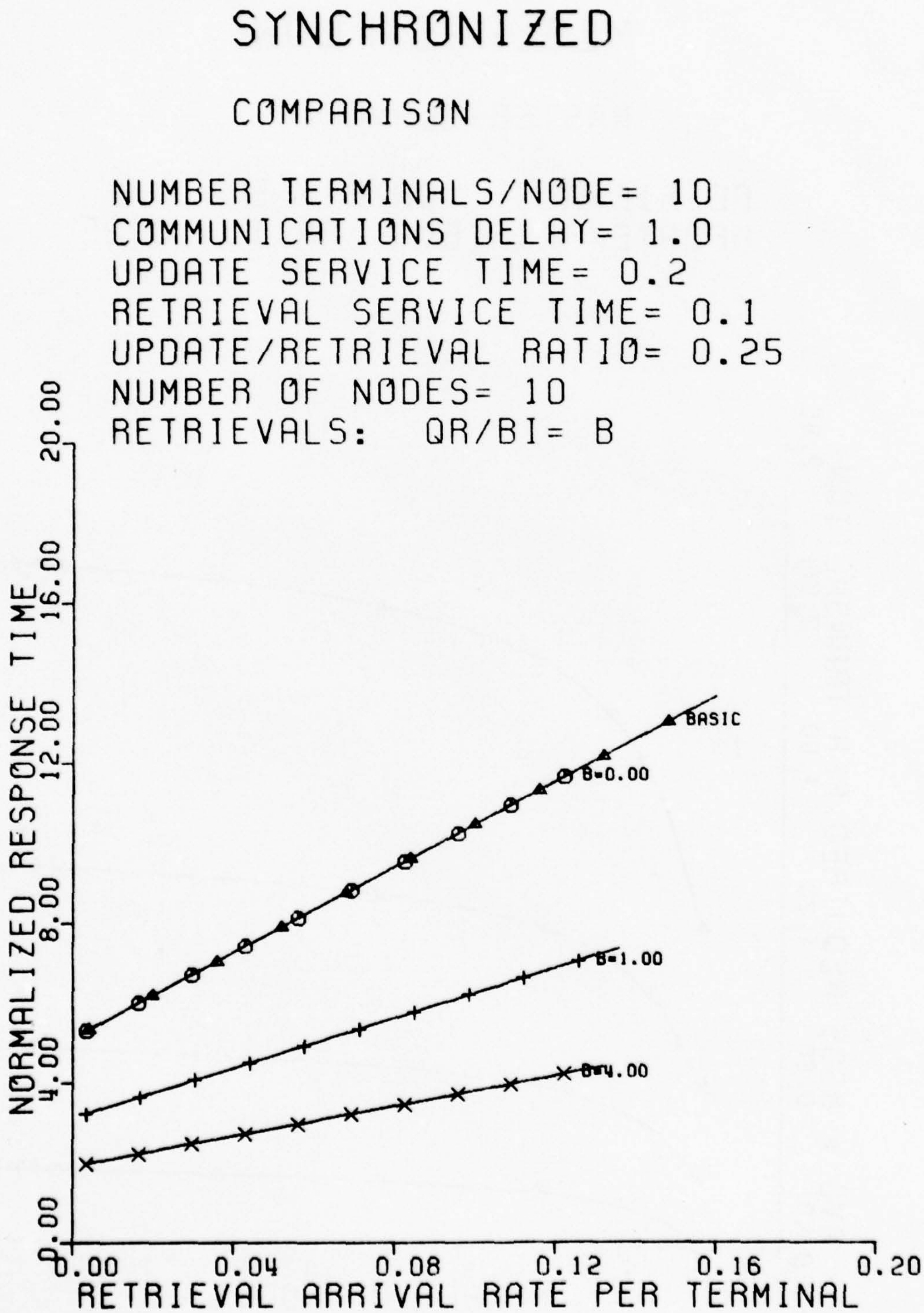


Figure 5-6. Performance Comparison for Synchronized Models



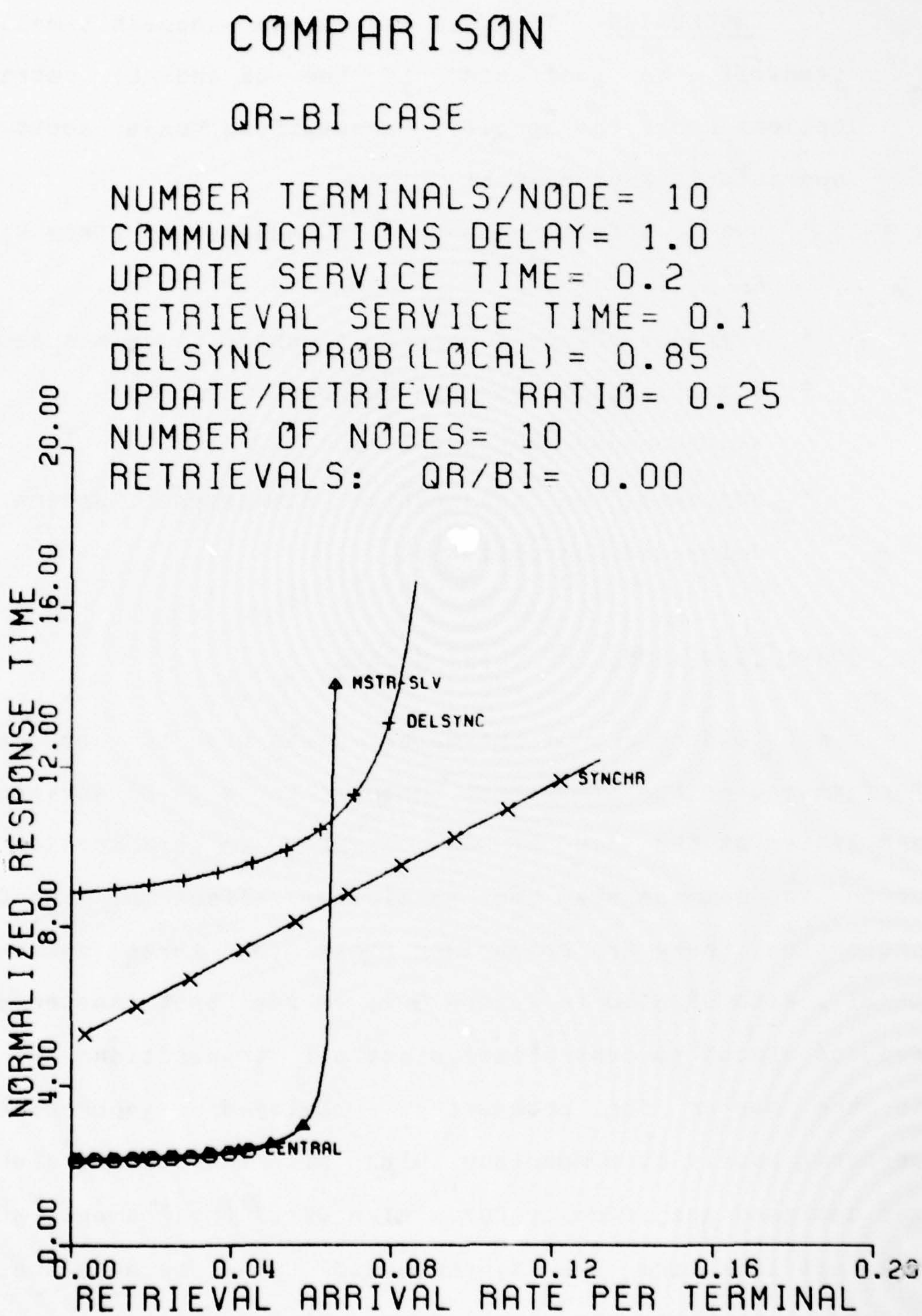
CONCLUSION: The data timeliness - access timeliness tradeoff, as manifested in the QR and BI retrieval options under the specific assumptions made about the operational environment:

* has no effect on centralized management (from Figure 4-4),
* degrades the performance of master/slave management,
* significantly improves the performance of synchronized management, and
* supports the alternative management scheme of delayed synchronization.

5.3 COMPARISON AMONG THE QR-BI MODELS

The standard set of parameter values used to compare the performance of the management schemes in a QR-BI environment are listed at the top of the figures in this section. In order to demonstrate the particular effect of the QR-BI proportion, there are comparison plots for three values of QR/BI. With QR/BI=0 in Figure 5-7, we see that master/slave reduces almost to centralized since all transactions are sent to the master for processing. Delayed synchronization performs poorly by comparison with all the others, since it was designed specifically for a high QR/BI environment, and it does not look like the synchronized case, because the (BI) retrieval requests are circulating, not the updates.

Figure 5-7. Performance Comparison of QR-BI Models for QR/BI=0



In Figure 5-8 with QR/BI=1, retrievals are equally divided between QR and BI types. Master/slave improves some over centralized because handling the QRs locally reduces contention at the master. Synchronized improves some because QRs may be processed as they arrive between batches. Delayed synchronization improves most since now only half the retrievals (BIs) are circulated, while nearly half (i.e., the local QRs) are handled locally. In fact, the delsync approach now competes quite well with the others.

The improvements for synch, delsync, and master/slave continue relative to the centralized case, Figure 5-9, as QR/BI continues to increase and the greater proportion of local activity reduces the effects of the communications delays on the average response time.

> Conclusion: Delayed synchronization is an appropriate management alternative offering significant performance improvement in an environment which has a high transaction arrival rate with a large proportion of quick response requests.

As in the basic case, the cost comparison among the management schemes is represented by the network message traffic requirements, Figures 5-10 and 5-11. Again, the average is computed over all types of transactions and the particular cases for the standard parameter set (10 nodes) are indicated by arrows. Figure 5-10 compares the schemes for a

Figure 5-8. Performance Comparison of QR-BI Models for QR/BI=1

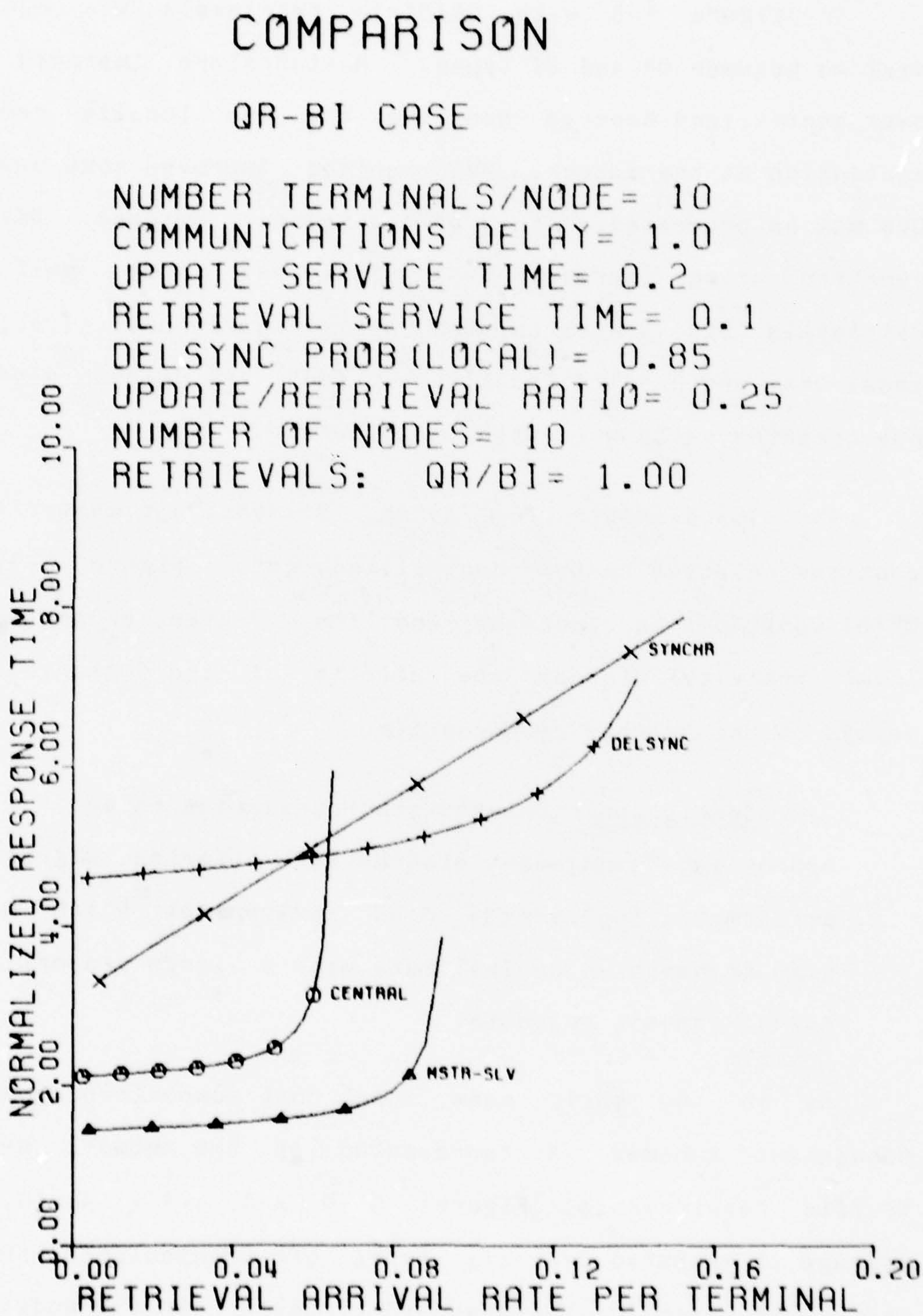


Figure 5-9. Performance Comparison of QR-BI Models for QR/BI=4

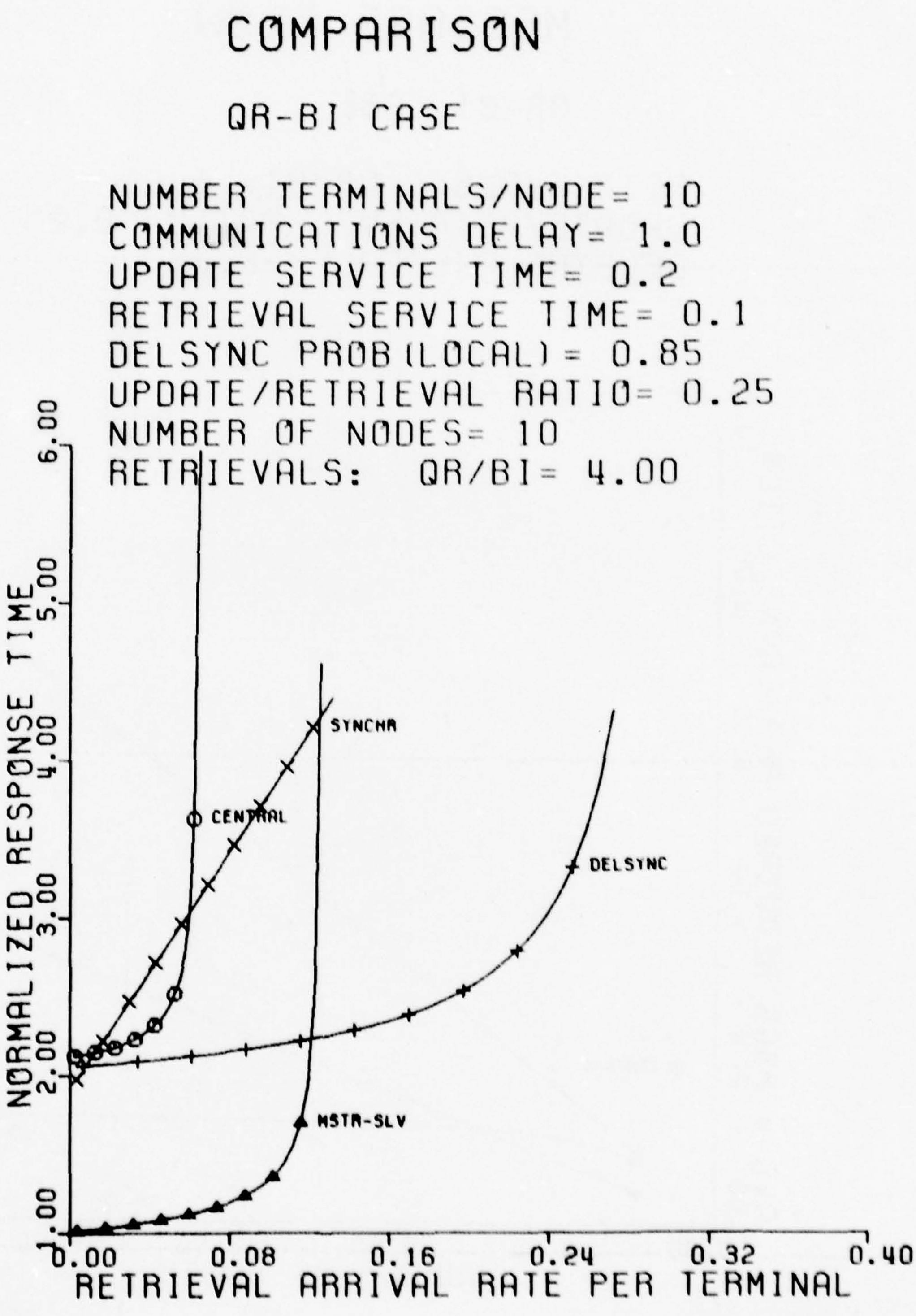
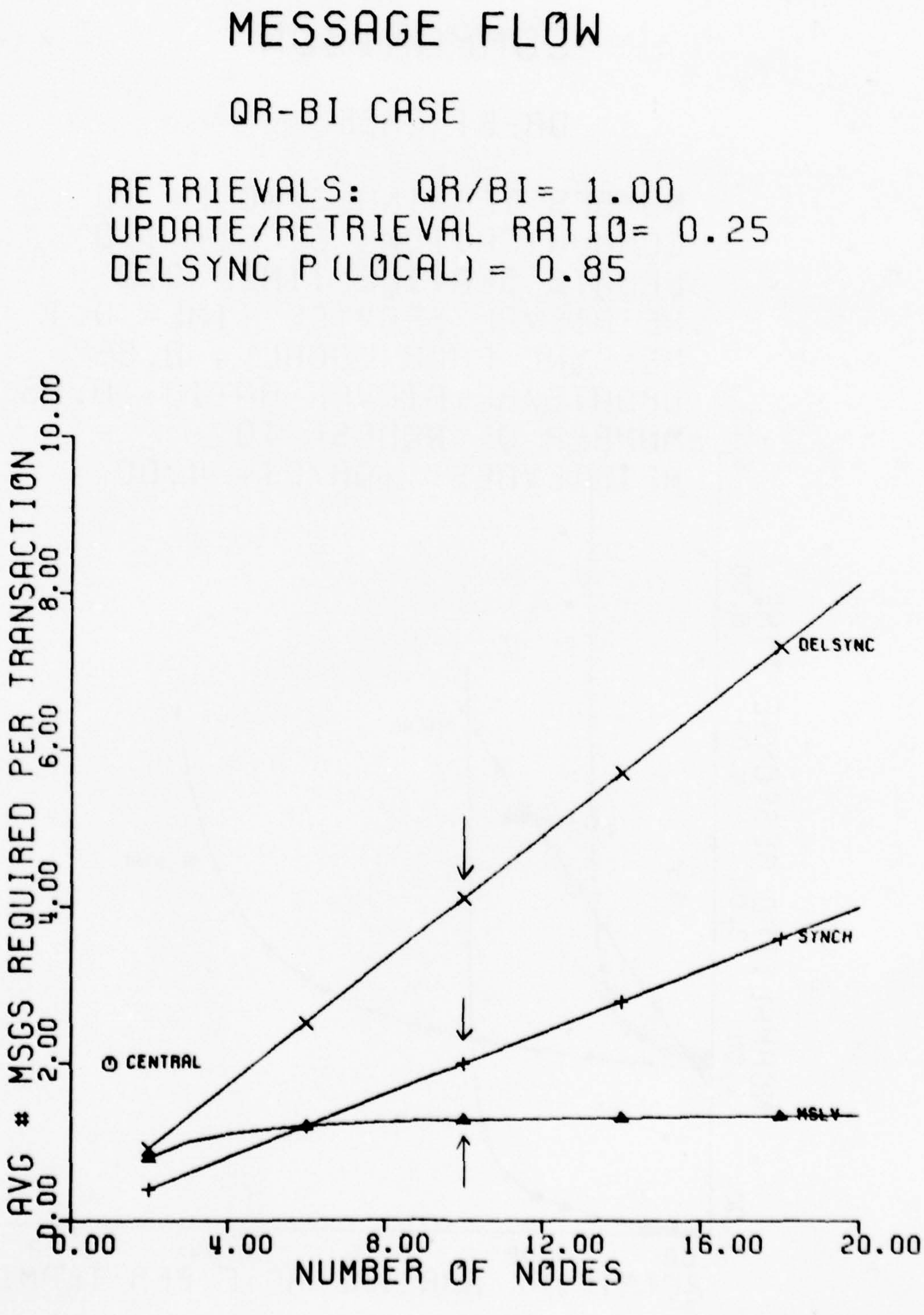


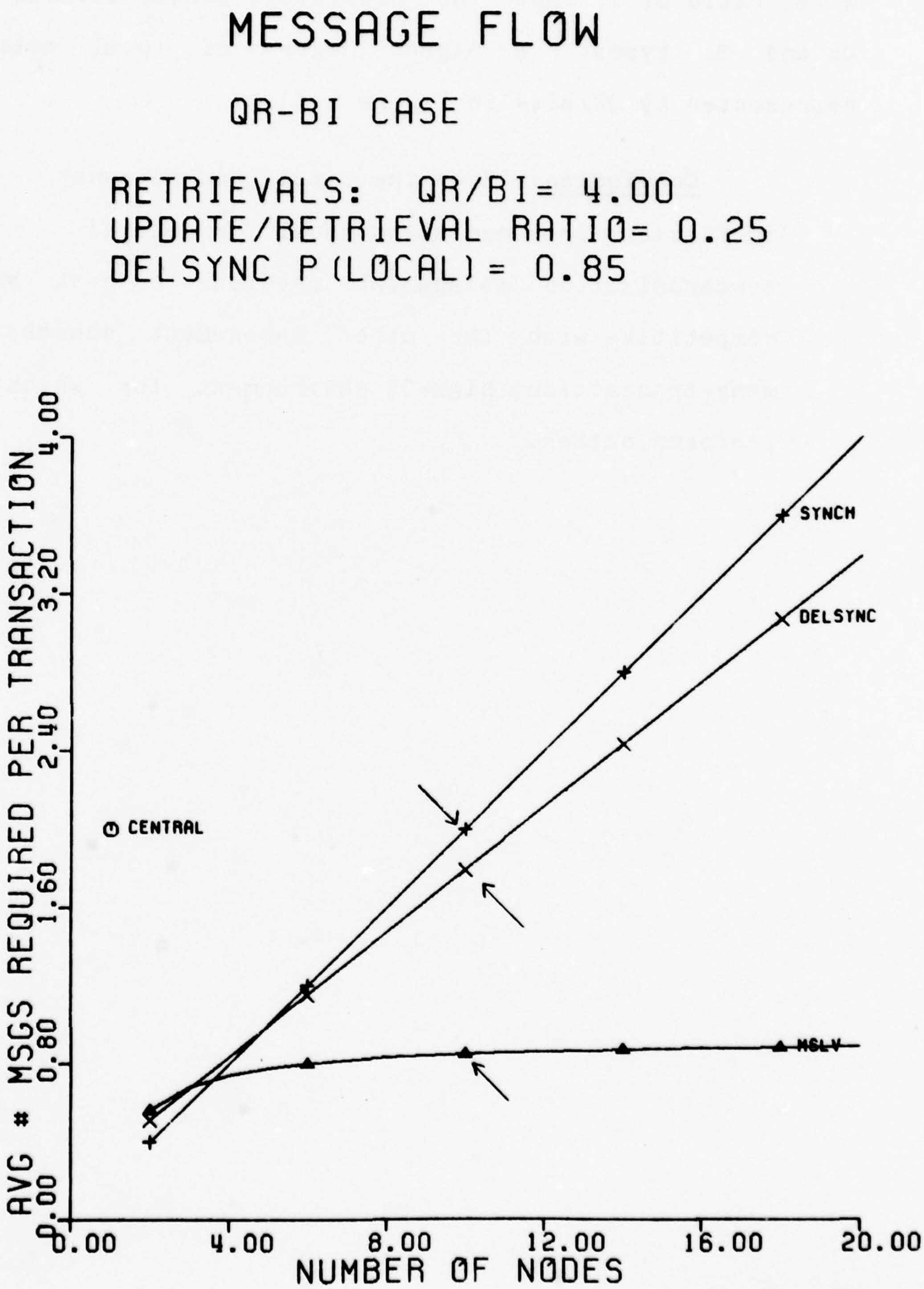
Figure 5-10. Cost Comparison of QR-BI Models for QR/BI=1



QR/BI ratio of 1, that is, retrievals evenly divided between QR and BI types. A higher degree of local activity is represented by QR/BI=4 in Figure 5-11.

<u>Conclusion</u>: Even the simple and somewhat inefficient implementation model for delayed synchronization management predicts a cost that is competitive with the other management schemes in the many-transaction, high-QR environment for which delsync performs better.

Figure 5-11. Cost Comparison of QR-BI Models for QR/BI=4



Chapter 6

CONCLUSIONS AND IMPLICATIONS

6.1 SUMMARY OF CONCLUSIONS

The evaluation methodology that has been developed is a valuable tool for comparing different management approaches to distributed database systems. The flow diagram technique emphasizes the importance of analyzing the system to identify both the specific control paths and the data flow that are required to process each type of database transaction. It is from such information that specific cost/performance results can be derived. Because few distributed database systems have been implemented and those mostly serve the special-purpose needs of particular applications, the cost/performance predictions of Chapter 5 remain to be verified:

* The virtual ring with token and tickets scheme for synchronized management is more appropriate for local networks than for long-distance, geographically dispersed networks.
* In the assumed operational environment, the quick response and best information retrieval options significantly improve the cost/performance of

synchronized management while providing no benefit to centralized management and actually degrading the cost/performance of master/slave management.

* Delayed synchronization management offers significant performance improvement at competitive costs for appropriate applications in an environment where the transaction arrival rate is high and the proportion of retrievals that are quick response requests is large.

Despite the many assumptions required in the modeling and the simple queuing approach of Poisson arrivals and exponential service distributions, the wide range of variation in the predictions provides very useful information to distinguish among the management schemes. Although the specific results are not at all surprising, in that they do not contradict our intuition about the management approaches, they do provide more formalism to the comparisons and they will allow even more detailed comparison when experimental data become available to verify the predictions.

6.2 SUGGESTIONS FOR FURTHER WORK

6.2.1 Modeling and Analysis

An extension to the analytical work which might be suggested is to give up the simple representation of communications delay as a constant and to choose a queuing

model. This would allow consideration of any extra delays in response time because of contention over the communications resources. However, increasing experience with broadcast contention networks like Ethernet and consideration of sparse loading conditions for packet networks tend to support the simpler assumption of constant delays.

A refinement of the models which would be useful is to include more detail about what activity occurs within each node (see [BUCC79], for example). Contributions from time spent in the central processor could be distinguished from input/output contributions, and the effects of multiprogramming and multiple storage units could be considered. It might also be useful to give up the assumption that all nodes are identical. Particularly in the case of master/slave management, the system performance would be improved by providing the master with faster or more powerful equipment while the percentage increase in the total system purchase price might not be very significant. A sample question which could be answered this way is how much faster would the master node have to be in order not to saturate before the individual slave nodes do.

A useful extension of the modeling would be to give up the assumption that transaction arrivals are uniformly distributed over all terminals of all nodes. This would allow analysis of clustered transaction arrivals against complete

duplicate copies of the database and against incomplete copies where the degree of locality of reference in the arrivals becomes important.

Adding results from such further work to the basic results already developed would probably give enough information to develop a table of performance predictions based on both the application characteristics and the management scheme as implemented on some specific network. As data become available to substantiate some of the predictions, the table might begin to offer guidelines for system designers to use in matching applications with management approaches. With such a goal in mind, it might be more appropriate to use the operational analysis approach to queuing network solutions initiated by Buzen (see particularly [DENN78]) since it is based on operational experience, rather than relying on the stochastic modeling that is often difficult to relate closely to real systems. Data gathered from experience with existing networks should begin to provide the parameter values which will be necessary to carry out the computations.

6.2.2 Data Activity Index

An interesting extension to the work of this dissertation (which is much smaller in scope than the above suggestions for modeling and analysis) is to pursue in more detail the mechanisms to support the quick response and best information

retrieval options as an explicit implementation of the timeliness tradeoff. The data quality indicator (DQI, see section 3.2.3) has potential for general use in multiple-copy distributed systems where there is any likelihood for retrieval accesses to be delayed due to higher priority traffic or operations. Availability of local data with quality information may well serve the immediate or temporary requirements of a variety of users.

Data quality indicators (section 3.2.3) could represent a crude indication of what the history of update activity on a particular data item had been, if there were some initial time reference point and a well-established or known function of time to describe the activity. We could consider adding to the DQI a timestamp for database insertion of the item, but the overhead of the DQI is already so high that its practicability is dubious. Instead, let us step up a level of granularity again, and consider files (as a grouping of items) rather than individual data items. Just as different DQIs apply to different types of data, so do different update patterns go with different types of data. If we could characterize those patterns, we could associate with each file a data activity index which would give more information to users about the data.

For example, text files begin with insertions, tend to have a high update rate for a while and then taper off as

revisions are made and a final version is approached. This might be approximated by an exponential distribution for the update frequency versus time; and not everyone handles his text files this way. So the areas to be considered within this topic would include:

* characterization of data types and update frequencies,
* appropriate parameters for characterizing the descriptive distributions, and
* what the collection of activity histories would entail.

It is interesting to notice that the technology of demand paging and automatic memory management already are involved with keeping track of data activity at a low level of sophistication. For example, there are counters or use bits referred to by procedures such as least recently used page removal algorithms. The data activity index is a sort of natural extension that is appropriate to a whole network context rather than to the memory hierarchy of a single machine.

6.3 SOME IMPLICATIONS

The specific results which have been presented to compare the cost/performance of the various management schemes are important primarily as examples of what can be done using the evaluation methodology. There are other proposals for both master/slave and synchronized management schemes that can be

evaluated; we chose the specific examples of n-host resiliency for a master/slave scheme and the virtual ring with token and tickets for a synchronized scheme. Furthermore, as new management proposals are made, they too can be evaluated, just as the delayed synchronization scheme was.

The evaluation methodology thus gives us an opportunity to lay out an entire spectrum of management strategies and investigate their appropriateness for particular application and network operating characteristics. The emphasis has been on the average system cost/performance; more detail in the queuing network model may lead to consideration of other factors.

In addition to this general technological impact, it will be interesting to see what use can be made of the specific proposal for a timeliness tradeoff. It may not be too difficult to augment some existing database management system to handle quick response and best information retrievals. This would offer experience that might be valuable in building synchronized distributed systems, where we have seen that the explicit tradeoff in the two retrieval types has the potential for really improving the average system cost/performance.

Chapter 7

REFERENCES AND BIBLIOGRAPHY

References   Bibliography

---------- ------------

[ALSB76] Alsberg, P.A. and J.D. Day, "A Principle for Resilient Sharing of Distributed Resources," Brown University Workshop in Distributed Processing, August 1976.

Alsberg, P.A., G.G. Belford, S.R. Bunch, J.D. Day, E. Grapa, D.C. Healy, E.J. McCauley and D.A. Willcox, Research in Network Data Management and Resource Sharing, Synchronization and Deadlock, Center for Advanced Computation Document Number 185, University of Illinois at Urbana-Champaign, 1 March 1976, 85 pp.

Alsberg, P.A., G.G. Belford, J.D. Day and E. Grapa, Research in Network Data Management Resource Sharing, Multi-Copy Resiliency Techniques, Center for Advanced Computation Document Number 202, University of Illinois at Urbana-Champaign, 31 May 1976, 52 pp.

[BADA78] Badal, D.Z. and G.J. Popek, "A Proposal for Distributed Concurrency Control for Partially Redundant Distributed Data Base Systems," Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, 29-31 August 1978, pp. 273-285.

Babic, G.A., M.T. Liu and R. Pardo, "A Performance Study of the Distributed Loop Computer Network (DLCN)," Proceedings of Computer Networking Symposium, 15 December 1977, pp. 66-75.

Baskett, F., J.H. Howard and J.T. Montague, "Task Communication in DEMOS," Proceedings of Sixth ACM Symposium on Operating System Principles, November 1977, pp. 23-31.

Becker, H.B., "Let's Put Information Networks into Perspective," Datamation, vol. 24, no. 3, March 1978, pp. 81-86.

Belford, G.G., Research in Network Data Management Resource Sharing, Optimization Problems in Distributed Data Management, Center for Advanced Computation Document Number 197, University of Illinois at Urbana-Champaign, 1 May 1976, 22 pp.

Berg, J.L. (ed), Data Base Directions: The Next Steps, ACM SIGMOD Record, vol. 8, no. 4,

November 1976, 158 pp.

Bernstein, P.A., D.W. Shipman, J.B. Rothnie and N. Goodman, The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (The General Case), Computer Corporation of America, CCA-77-09, 15 December 1977, 172 pp.

[BERN78] Bernstein, P.A. and D.W. Shipman, "A Formal Model of Concurrency Control Mechanisms for Database Systems," Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, 29-31 August 1978, pp. 189-205.

Bonczek, R.H., C.W. Holsapple and A.B. Whinston, Information Transferral Within a Distributed Data Base Via a Generalized Mapping Language, Krannert Graduate School of Management Paper Number 577, Purdue University, 1 November 1976, 20 pp.

Booth, G.M., "The Use of Distributed Data Bases in Information Networks," Computer Communication Impacts and Implications, S. Winkler (ed), Proceedings First International Conference on Computer Communication, Washington, DC, 24-26 October 1972, pp. 371-376.

[BUCC79] Bucci, G. and D.N. Streeter, "A Methodology for the Design of Distributed Information Systems," Communications of the ACM, vol. 22, no. 4, April 1979, pp. 233-244.

Buzen, J.P., "Fundamental Operational Laws of Computer System Performance," Acta Informatica, vol. 7, 1976, pp. 167-182.

Buzen, J.P., "Operational Analysis: An Alternative to Stochastic Modeling," Performance of Computer Installations (Conference Proceedings), North-Holland, June 1978, pp. 175-194.

Casey, L. and N. Shelness, "A Domain Structure for Distributed Computer Systems," Proceedings of the Sixth Symposium on Operating Systems Principles, Operating Systems Review, vol. 11, no. 5, November 1977, pp. 101-108.

Casey, R.G., "Allocation of Copies of Files in an Information Network," Proceedings AFIPS 1972 Spring Joint Computer Conference, vol. 40, AFIPS Press, Montvale, NJ, pp. 617-625.

[CHAN76] Chandy, K.M. and J.E. Hewes, "File Allocation in Distributed Systems," Proceedings of the International Symposium on Computer Performance

Modeling, Measurement and Evaluation, 29-31 March 1976, P.P.S. Chen and M. Franklin (eds), ACM, 1976, pp. 10-13.

Chang, E., "A Class of Decentralized Mechanisms for Mutual Exclusion in Distributed Systems," University of Toronto, 16 pp.

Chang, E., "Decentralized Deadlock Detection in Distributed Systems," University of Toronto, 17 pp.

[CHUW73] Chu, W.W., "Optimal File Allocation in a Computer Network," Computer-Communication Networks, N. Abramson and F.F. Kuo (eds), Prentice-Hall, Englewood Cliffs, New Jersey, 1973, pp. 82-94.

Chu, W.W., "Optimal File Allocation in a Multiple Computer System," IEEE Transactions on Computers, vol. C-18, no. 10, October 1969, pp. 885-889.

Chu, W.W., "Performance of File Directory Systems for Data Bases in Star and Distributed Networks," Proceedings National Computer Conference, 1976, pp. 577-587.

Chu, W.W. and E.E. Nahouraii, "File Directory Design Considerations for Distributed Data Bases," Proceedings of the International Conference on

Very Large Data Bases, D.S. Kerr (ed), ACM, 1975, pp. 543-545.

Chu, W.W. and G. Ohlmacher, "Avoiding Deadlock in Distributed Data Bases," Proceedings ACM National Symposium, vol. 1, March 1974, pp. 156-160.

Comba, P.G., "Needed: Distributed Control," Proceedings of the International Conference on Very Large Data Bases, D.S. Kerr (ed), ACM, 1975, pp. 364-375.

Crocker, T.H. and D.M. Klamer, Models of Optimal File Allocation in a Distributed Data Base: A Survey, Naval Ocean Systems Center, Technical Report 195, 15 January 1978, 25 pp.

Curtice, R.M., "The Outlook for Data Base Management," Datamation, April, 1976, pp. 46-49.

[DATE75] Date, C.J., An Introduction to Database Systems, Addison-Wesley, 1975.

[DAVI73] Davies, D.W. and D.L.A. Barber, Communication Networks for Computers, John Wiley and Sons, 1973.

Day, J.D. and G.G. Belford, A Cost Model for Data Distribution, Center for Advanced Computation Document Number 179, University of Illinois at Urbana-Champaign, 1 November 1975.

Denning, D.E., "A Lattice Model of Secure Information Flow," <u>Communications of the ACM</u>, vol. 19, no. 5, May 1976, pp. 236-243.

[DENN76] Denning, P.J., "Fault Tolerant Operating Systems," <u>Computing Surveys</u>, vol. 8, no. 4, December 1976, pp. 359-389.

[DENN78] Denning, P.J. and J.P. Buzen, "The Operational Analysis of Queueing Network Models," <u>Computing Surveys</u>, vol. 10, no. 3, September 1978, pp. 225-261.

Dijkstra, E.W., "Self-Stabilizing Systems in Spite of Distributed Control," <u>Communications of the ACM</u>, vol. 17, no. 11, November 1974, pp. 643-644.

<u>A Distributed Database Management System for Command and Control Applications, Semi-Annual Technical Report 1</u>, Computer Corporation of America, CCA-77-06, 30 July 1977, 125 pp.

[CCA 78] <u>A Distributed Database Management System for Command and Control Applications: Semi-Annual Technical Report 2</u>, Computer Corporation of America, CCA-78-03, 30 January 1978, 155 pp.

Elam, J. and J. Stutz, Some Considerations and Models for the Distribution of a Data Base, Center for Cybernetic Studies Research Report CCS 279, University of Texas, May 1976, 32 pp.

[ELLI77] Ellis, C.A., "Consistency and Correctness of Duplicate Database Systems," Proceedings of the Sixth Symposium on Operating Systems Principles, Operating Systems Review, vol. 11, no. 5, November 1977, pp. 67-84.

Eswaran, K.P., J.N. Gray, R.A. Lorie, and I.L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System," Communications of the ACM, vol. 19, November 1976, pp. 624-633.

Fabry, R.S., "Capability-Based Addressing," Communications of the ACM, vol. 17, no. 7, July 1974, pp. 403-412.

Farber, D.J., "Distributed Data Bases -- an Exploration," Computer Networks -- Trends and Applications, Symposium: 18 June 1975.

Farber, D.J., "Software Considerations in Distributed Architectures," Computer, vol. 7, no. 3, March 1974, pp. 31-35.

Farber, D.J. and F.R. Heinrich, "The Structure of a Distributed Computer System -- The Distributed File System," Proceedings ICCC, October 1972, pp. 364-370.

Foley, J.D. and E.H. Brownlee, "A Model of Distributed Processing in Computer Networks, with Application to Satellite Graphics," Proceedings International Conference on Computer Communications, Stockholm, 1974.

Foley, J.D. and J.W. McInroy, "An Event-Driven Data Collection and Analysis Facility for a Two-Computer Network," 1974 SIGMETRICS Symposium on Measurement and Evaluation, Montreal, 1974.

Fredericksen, D.H., "Describing Data in Computer Networks," IBM Systems Journal, vol. 12, no. 3, 1973, pp. 257-282.

Garcia-Molina, H., "Performance Comparison of Two Update Algorithms for Distributed Databases," Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, 29-31 August 1978, pp. 108-119.

[GARC78] Garcia-Molina, H., Performance Comparison of Update Algorithms for Distributed Databases, Progress Report # 1, Stanford University, 1978.

Garcia-Molina, H., Performance Comparison of Update Algorithms for Distributed Databases, Progress Report # 2, Stanford University, 1978.

[GELE78] Gelenbe, E. and K. Sevcik, "Analysis of Update Synchronization for Multiple Copy Data-Bases," Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, 29-31 August 1978, pp. 69-90.

[GERR77] Gerritsen, R., H. Morgan, and M. Zisman, "On Some Metrics for Databases or What is a Very Large Database?" ACM SIGMOD Record, vol. 9, no. 1, June 1977, pp. 50-74.

Ghosh, S.P., "Distributing a Data Base with Logical Associations on a Computer Network for Parallel Searching," IEEE Transactions on Software Engineering, vol. SE-2, no. 2, June 1976, pp. 106-113.

Ghosh, S.P. and W.G. Tuel, Jr., "A Design of an Experiment to Model Data Base System Performance," IEEE Transactions on Software Engineering, vol. SE-2, no. 2, June 1976, pp. 97-106.

Gitman, I. and H. Frank, "Economic Analysis of Integrated Voice and Data Networks: A Case

Study," Proceedings of the IEEE, vol. 66, no. 11, November 1978, pp. 1549-1570.

Gouda, M.G., "A Hierarchical Controller for Concurrent Accessing of Distributed Databases," Fourth Workshop on Computer Architecture for Non-Numeric Processing, 1-4 August 1978 (SIGMOD vol. X, no. 1), pp. 65-70.

[GRAP76] Grapa, E., Characterization of a Distributed Data Base System, Department of Computer Science, UIUCDCS-R-76-831, University of Illinois at Urbana-Champaign, October 1976.

Grapa, E. and G.G. Belford, "Some Theorems to Aid in Solving the File Allocation Problem," Communications of the ACM, vol. 20, no. 11, November 1977, pp. 878-882.

Healy, D.C., E.J. McCauley and D.A. Willcox, Research in Network Data Management and Resource Sharing, Experimental System Progress Report, Center for Advanced Computation Document Number 209, University of Illinois at Urbana-Champaign, 30 September 1976, 81 pp.

Hevner, A.R. and S.B. Yao, "Query Processing on a Distributed Database," Proceedings of the Third Berkeley Workshop on Distributed Data Management

and Computer Networks, 29-31 August 1978, pp. 91-107.

Holler, E. and O. Drobnik, "Implementation of Decentralized Coordination Mechanisms in Distributed Mini-/Micro-computer Systems," Brown University Workshop in Distributed Processing, August 1977.

Hsiao, D.K. and K. Kannan, The Architecture of a Database Computer, Part II, The Design of Structure Memory and its Related Processors, Computer and Information Science Research Center Report Number OSU-CISRC-TR-76-2, The Ohio State University, October 1976, 113 pp.

[IBM 77] International Business Machines, "DP Dialogue; J.I. Case: Super Service on Spare Parts," advertisement in Mini-Micro Systems, April 1977.

Jensen, E.D. and W.E. Boebert, "Partitioning and Assignment of Distributed Software," Brown University Workshop in Distributed Processing, August 1976.

[JOHN75] Johnson, P.R. and R.H. Thomas, "The Maintenance of Duplicate Databases," RFC #677, NIC #31507, January 1975.

[KLEI75] Kleinrock, L., Queueing Systems, Volume I: Theory, John Wiley & Sons, 1975.

[KLEI76] Kleinrock, L., Queueing Systems, Volume II: Computer Applications, John Wiley & Sons, 1976.

Kimbleton, S.R., "A Fast Approach to Network Data Assignment," Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks, 25-27 May 1977, pp. 245-256.

Kimbleton, S.R. and G.M. Schneider, "Computer Communication Networks: Approaches, Objectives, and Performance Considerations," Computing Surveys, vol. 7, no. 3, September 1975, pp. 129-173.

Labetoulle, J., E.G. Manning and R.W. Peebles, Analysis and Simulation of a Homogeneous Computer Network, Computer Communications Network Group, Report E-30, University of Waterloo, January 1975, 49 pp.

Lamport, L., "Concurrent Reading and Writing," Communications of the ACM, vol. 20, no. 11, November 1977, pp. 806-811.

[LAMP78] Lamport, L., Time, Clocks, and the Ordering of Events in a Distributed System, Massachusetts

Computer Associates, Inc., March 1976, and in Communications of the ACM, vol. 21, no. 7, July 1978, pp. 558-565.

[LAMB76] Lampson, B. and H. Sturgis, "Crash Recovery in a Distributed Data Storage System," Brown University Workshop in Distributed Processing, August 1976.

[LELA78] LeLann, G., "Algorithms for Distributed Data-Sharing Systems Which Use Tickets," Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, 29-31 August 1978, pp. 259-272.

[LELA77] LeLann, G., "Distributed Systems -- Towards a Formal Approach," IFIP Congress '77, Toronto, Canada, August 1977.

[LEVI76] Levin, K.D., Organizing Distributed Data Bases in Computer Networks, Wharton School, University of Pennsylvania, September 1976.

Levin, K.D. and H.L. Morgan, Dynamic File Assignment in Computer Networks Under Varying Access Request Patterns, Decision Sciences Department Report Number 75-04-01, Wharton School, University of Pennsylvania, April 1975, 21 pp.

Levin, K.D. and H.L. Morgan, Optimizing

Distributed Data Bases - A Framework for Research, Decision Sciences Department Report Number 75-01-07, Wharton School, University of Pennsylvania, January 1975, 6 pp.

[LEVI77] Levine, P.H., _Facilitating Interprocessor Communication in a Heterogeneous Network Environment_, Laboratory for Computer Science, MIT/LCS/TR-184, Massachusetts Institute of Technology, July 1977.

Linden, T.A., "Operating System Structures to Support Security and Reliable Software," _Computing Surveys_, vol. 8, no. 4, December 1976, pp. 409-445.

Maekawa, M., "An Extensible Distributed Data Base System," Brown University Workshop in Distributed Processing, August 1977.

[MAHM76] Mahmoud, S. and J.S. Riordon, "Optimal Allocation of Resources in Distributed Information Networks," _ACM Transactions on Database Systems_, vol. 1, no. 1, March 1976, pp. 66-78.

Mahmoud, S. and J.S. Riordon, "Protocol Considerations for Software Controlled Access Methods in Distributed Data Bases," _Proceedings of the International Symposium on Computer_

Performance Modeling, Measurement and Evaluation, P.P.S. Chen and M. Franklin (eds), ACM, 1976, pp. 241-264.

Manacher, G.K., "On the Feasibility of Implementing a Large Relational Data Base with Optimal Performance on a Minicomputer," Proceedings of the International Conference on Very Large Data Bases, D.S. Kerr (ed), ACM, 1975, pp. 175-201.

[MAND78] Mandell, M., "Distributed Data Processing," Computer Decisions, vol. 10, no. 7, July 1978, pp. 24-28.

Manning, E. and R.W. Peebles, A Homogeneous Network for Data Sharing -- Communications, Computer Communications Network Group, Report E-12, University of Waterloo, March 1974, 63 pp.

Martin, J., Computer Data-Base Organization, Prentice-Hall, 1975.

[MART76] Martin, J., Principles of Data-Base Management, Prentice-Hall, 1976.

Maryanski, F.J., "A Survey of Developments in Distributed Data Base Management Systems," Computer, vol. 11, no. 2, February 1978,

pp. 28-38.

[MCEN75] McEnroe, P.V., H.T. Huth, E.A. Moore and W.W. Morris, "Overview of the Supermarket System and the Retail Store System," IBM Systems Journal, vol. 14, no. 1, 1975, pp. 3-15.

Menasce, D.A. and R.R. Muntz, "Locking and Deadlock Detection in Distributed Databases," Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, 29-31 August 1978, pp. 215-232.

Millen, J.K., "Security Kernel Validation in Practice," Communications of the ACM, vol. 19, no. 5, May 1976, p. 243.

Miller, T.J., Deadlock in Distributed Computer Systems, Department of Computer Science, UIUCDCS-R-74-619, University of Illinois at Urbana-Champaign, December 1974, 57 pp.

Minsky, N., "Intentional Resolution of Privacy Protection in Database Systems," Communications of the ACM, vol. 19, no. 3, March 1976, pp. 148-159.

[MORG77] Morgan, H.L. and K.D. Levin, "Optimal Program and Data Locations in Computer Networks," Communications of the ACM, vol. 20, no. 5, May

1977, pp. 315-322.

Morris, P. and D. Sagalowicz, "Managing Network Access to a Distributed Database," Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks, 25-27 May 1977, pp. 58-67.

Needham, R.M., "The CAP Project -- An Interim Evaluation," Proceedings of the Sixth Symposium on Operating Systems Principles, Operating Systems Review, vol. 11, no. 5, November 1977, pp. 17-22.

Needham, R.M. and A.D. Birrell, "The CAP Filing System," Proceedings of the Sixth Symposium on Operating Systems Principles, Operating Systems Review, vol. 11, no. 5, November 1977, pp. 11-16.

Needham, R.M. and R.D.H. Walker, "The Cambridge CAP Computer and its Protection System," Proceedings of the Sixth Symposium on Operating Systems Principles, Operating Systems Review, vol. 11, no. 5, November 1977, pp. 1-10.

Papadimitriou, C.H., Serializability of Concurrent Updates, Center for Research in Computing Technology TR-14-78, Harvard University, 51 pp.

Pardo, R., M.T. Liu and G.A. Babic, "Distributed Services in Computer Networks: Designing the Distributed Loop Data Base System (DLDBS)," Proceedings of Computer Networking Symposium, 15 December 1977, pp. 60-65.

Pardo, R., M.T. Liu and G.A. Babic, "An N-Process Communication Protocol for Distributed Processing," Computer Network Protocols, A. Danthine (ed), Universite de Liege, 1978, pp. D7-1 to D7-10.

Passafiume, J.J. and S. Wecker, "Distributed File Access in DECnet," Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks, 25-27 May 1977, pp. 114-129.

[PEEB77] Peebles, R.W., "Concurrent Access Control in a Distributed Transaction Processing System," Brown University Workshop in Distributed Processing, August 1977.

Peebles, R.W., Design Considerations for a Distributed Data Access System, Wharton School, University of Pennsylvania, 1973.

Peebles, R.W. and E. Manning, "A Computer

Architecture for Large (Distributed) Data Bases," Proceedings of the International Conference on Very Large Data Bases, D.S. Kerr (ed), ACM, 1975, pp. 405-427.

Peebles, R.W. and E. Manning, "System Architecture for Distributed Data Management," Computer, vol. 11, no. 1, January, 1978, pp. 40-47.

Pliner, M., L. McGowan and K. Spalding, "A Distributed Data Management System for Real-Time Applications," Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks, 25-27 May 1977, pp. 68-86.

Pouzin, L., "CIGALE, The Packet Switching Machine of the CYCLADES Computer Network," Information Processing 74, North-Holland, 1974, pp. 155-159.

[REED78] Reed, D.P., Naming and Synchronization in a Decentralized Computer System, Laboratory for Computer Science MIT/LCS/TR-205, Massachusetts Institute of Technology, September 1978, 181 pp.

Reed, D.P. and R.K. Kanodia, "Synchronization with Eventcounts and Sequencers," Communications of the ACM, vol. 22, no. 2, pp. 115-123.

Research in Network Data Management and Resource

Sharing -- Final Research Report, Center for Advanced Computation Document Number 210, University of Illinois at Urbana-Champaign, 30 September 1976, 55 pp.

Rodriguez-Rosell, J., "Empirical Data Reference Behavior in Data Base Systems," Computer, vol. 9, no. 11, November 1976, pp. 9-13.

Rosenkrantz, D.J., R.E. Stearns and P.M. Lewis, "A System Level Concurrency Control for Distributed Database Systems," Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks, 25-27 May 1977, pp. 132-145.

[ROSE78] Rosenkrantz, D.J., R.E. Stearns and P.M. Lewis, "System Level Concurrency Control for Distributed Database Systems," ACM Transactions on Database Systems, vol. 3, no. 2, June 1978, pp. 178-198.

Rothnie, J.B. and N. Goodman, An Approach to Updating in a Redundant Distributed Data Base Environment, Computer Corporation of America, CCA-77-01, 15 February 1977, 99 pp.

[ROTH77] Rothnie, J.B. and N. Goodman, "An Overview of the Preliminary Design of SDD-1: A System for Distributed Databases," Proceedings of the Second

Berkeley Workshop on Distributed Data Management and Computer Networks, 25-27 May 1977, pp. 39-57.

Rzepka, W.E., Considerations in the Design of a Secure Data Base Management System, Rome Air Development Center, RADC-TR-77-9, March 1977, 26 pp.

Saltzer, J.H., "Research Problems of Decentralized Systems with Largely Autonomous Nodes," ACM Operating Systems Review, vol. 12, no. 1, January 1978, pp. 43-52.

[SENK77] Senko, M.E., "Data Structures and Data Accessing in Data Base Systems Past, Present, Future," IBM Systems Journal, vol. 16, no. 3, 1977, pp. 208-257.

[SEVE76] Severance, D.G. and G.M. Lohman, "Differential Files: Their Application to the Maintenance of Large Databases," ACM Transactions on Database Systems, vol. 1, no. 3, September 1976, pp. 256-267.

[SHAP77] Shapiro, R.M. and R.E. Millstein, NSW Reliability Plan, Massachusetts Computer Associates, CA-7707-0511 (draft), 5 July 1977, 146 pp.

Shoshani A., "Data Sharing in Computer Networks,"

WESCON Conference, September 1972.

Sockut, G.H., Data Base Performance Under Concurrent Reorganization and Usage, Center for Research in Computing Technology TR-12-77, Harvard University, July 1977, 166 pp.

[SOLO78] Solomon, R.J., "Packet Networks," Mini-Micro Systems, vol. 11, no. 3, March 1978, pp. 48-53.

Stearns, R.E., P.M. Lewis II and D.J. Rosenkrantz, "Concurrency Control for Database Systems," Proceedings 17th Annual Symposium on Foundations of Computer Science, IEEE, 1976, pp. 19-32.

Stone, H.S., Multiprocessor Scheduling with the Aid of Network Flow Algorithms, Department of Electrical and Computer Engineering, University of Massachusetts, November 1975, 41 pp.

[STON78] Stonebraker, M., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES," Proceedings of the Third Berkeley Workshop on Distributed Data Management and Computer Networks, 29-31 August 1978, pp. 235-258.

[STON76] Stonebraker, M., E. Wong, P. Kreps, and G. Held, "The Design and Implementation of INGRES," ACM

Transactions on Database Systems, vol. 1, no. 3, September 1976, pp. 189-222.

[STON77] Stonebraker, M. and E. Neuhold, "A Distributed Database Version of INGRES," Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks, 25-27 May 1977, pp. 19-36.

Streeter, D.N., "Centralization or Dispersion of Computing Facilities," IBM Systems Journal, vol. 13, no. 3, 1973, pp. 283-301.

Stucki, M.J., J.R. Cox, Jr., G.-C. Roman and P.N. Turcu, "Coordinating Concurrent Access in a Distributed Database Architecture," Fourth Workshop on Computer Architecture for Non-Numeric Processing, 1-4 August 1978 (SIGMOD vol. X, no. 1), pp. 60-64.

[SUNS77] Sunshine, C.A., "Interconnection of Computer Systems to Enhance Availability," Brown University Workshop in Distributed Processing, August 1977.

Tajibnapis, W.D., "A Correctness Proof of a Topology Information Maintenance Protocol for a Distributed Computer Network," Communications of the ACM, vol. 20, no. 7, July 1977, pp. 477-485.

[THOM76] Thomas, R.H., "A Solution to the Update Problem for Multiple Copy Data Bases Which Uses Distributed Control," Brown University Workshop in Distributed Processing, August 1976.

Tobagi, F.A., M. Gerla, R.W. Peebles and E.G. Manning, "Modeling and Measurement Techniques in Packet Communication Networks," Proceedings of the IEEE, vol. 66, no. 11, November 1978, pp. 1423-1447.

[VERH78] Verhofstad, J.S.M., "Recovery Techniques for Database Systems," ACM Computing Surveys, vol. 10, no. 2, June 1978, pp. 167-195.

[WATK77] Watkins, J., telephone conversation, April 1977.

Wetterau, J.B., "Planning and Design Problems in a Distributed Data Base System for Loop Administration," Trends and Applications: 1978, Distributed Processing, IEEE, 1978, pp. 116-124.

[WONG77] Wong, E., "Retrieving Dispersed Data from SDD-1: A System for Distributed Databases," Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks, 25-27 May 1977, pp. 217-235.

[WYLE79] Wyleczuk, R.H., Timestamps and Capability-Based

Protection in a Distributed Computer Facility, Massachusetts Institute of Technology, February 1979, 122 pp.

Appendix A

DISTRIBUTED DATABASE RESEARCH

A.1 INTRODUCTION

Managing a distributed database begins with all of the problems involved in managing a centralized database and is complicated by having the management responsibility and the database itself divided among multiple computer systems which may be geographically as well as physically separate. The first purpose of this appendix is to outline those aspects of the problems which are specifically associated with the distributed nature of the data and its management. The second purpose is to briefly survey the work that has been done to date toward solving some of these problems so that real DDBMSs could be built.

The discussion is divided into problem areas which can be briefly characterized as:

<u>integrity:</u> correctness of individual data items in the context of the whole database as a model representing some enterprise, where a <u>data item</u> is the smallest independently accessible unit of data;

<u>organization:</u> location and arrangement of data and directories to facilitate efficient responses to user

requests;

security: protection of data from accidental or malicious corruption and prevention of unauthorized access;

data incompatibility: limitations on data usage because of its structure or representation;

reliability: availability, failure recovery, explicit indications of database status with regard to individual operations, and prevention or reduction of situations where data is inaccessible; and

implementation experience in building systems for actual use.

A difficulty often encountered in discussions of distributed databases is the lack of an agreed-upon vocabulary. For the purposes of this discussion, a database transaction is the general term for any user interaction with the database (DB). Transactions requiring only reads from the DB are called retrievals; transactions requiring writes are called updates (including insertions and deletions). In general, an update will require reading something before the writing is done. Transactions will involve some specific (set of) data item(s), the smallest accessible unit(s) of the database. Each data item consists of a value and a timestamp (node identification and clock time) for insertion or last update of the value. The granularity or what size a data item actually is will not be discussed except in terms of storage costs.

With this background, let us proceed to examine some problems. The discussions are intended to include: definition of the problem from a centralized point of view, additional considerations or complications in the distributed environment, a brief presentation of some proposed solutions, and mention of known restrictions or limitations of the solutions. In general, the choice of solutions discussed is based on details which were referred to in the body of the dissertation.

A.2 INTEGRITY

In the general context of database management, the problem of integrity includes both: (1) protection of data items from loss or destruction by errors and (2) assurance that data values are correct within the framework of the whole database as a model representing some enterprise (a further discussion can be found in Chapter 20 of [DATE75]). In the face of no really standard terminology, for the purposes of this thesis (1) above will be referred to as the problem of back-up and recovery and (2) above will be consistency. These problems, and their solutions, tend to interact in practice and the terminology is often intertwined and interchanged.

A distributed DBMS must be concerned with the same integrity problems as a centralized system and also deal with certain complications caused by the distribution. In

particular, back-up and recovery must handle a number of separate sites connected by some communications media. If communications fail, for example, the recovery algorithms may not be able to get to the back-up data they need or even decide what data are relevant. In addition, consistency must handle much larger (than centralized) communication delays and multiple copies of data. This may mean that without constant overhead communications such as "I'm okay, are you okay?", it will be impossible for any node to differentiate between failure of another node and extra-long communications delays. If timeout mechanisms are used, then the possibility of duplicate messages must be handled (e.g., at time-out a failure decision is reached and a retry is requested, and then the original result shows up). Simple extension of centralized database control strategies, such as locking data items to provide mutual exclusion or using pointers instead of redundancy, may not be appropriate across physically separate computer systems which may also be geographically dispersed.

For example, distributed systems without centralized control deliberately give up the ability to centrally control the total system state. Since the nodes of the system cooperate in control and since communication delays between any pair of nodes are unpredictable, there is no time reference common among the nodes and no global state information which can be collected that reflects the state of the whole system at a given time [LELA77]. This complicates

particular database management strategies commonly relied on for control in centralized systems. Centralized control of locking, for example, is often used to prevent interfering accesses to data items being referred to by multiple users. When lock control is distributed among dispersed nodes, the communications delays in transferring status information make it impossible for any node to collect the simultaneously accurate global state data needed to detect or resolve deadlock.

Another problem is that no two nodes of a general distributed system can be proven [LELA77] to identically observe the same sequence of events (e.g., internal state changes). The events are observable only through the external communications generated, and the delay time involved in receiving such communications is comparable to the time between the successive events. If observations cannot even be proven the same for a given event sequence, it will be difficult to ensure the same execution sequences (as for update applications) at different nodes. This is even further complicated, since no two nodes can be proven [LELA77] to have the same global view of even a particular system subset.

A.2.1 Back-up and Recovery

A variety of back-up and recovery techniques (for centralized systems) have been discussed in [VERH78] from the

point of view that a failure is "an event at which the system does not perform according to specifications" and where an error is "a piece of information which can cause a failure." The object of the recovery techniques is to restore the database to a state which is useable within the system, by using back-up information which has been collected during system operation.

In a distributed system, the previously mentioned communication delays and lack of global state information immediately complicate recovery solutions. If a single node fails and the rest of the system continues to operate, there must be a way for a recovering node to catch up on everything that has happened since it failed. The back-up information must have been kept in a way and in places that facilitate such recovery.

Crash recovery has been looked at by Lampson and Sturgis [LAMB76] in the context of "atomic transactions." A transaction is simply a user-designated sequence of read and write commands to be sent to the file system. The atomic property for transactions is aimed at ensuring that after recovery from a system crash, either all of the writes have been executed or none have. The technique for accomplishing this assumes that the processes which write on the system storage media can guarantee only three possible outcomes for any write action:

1. it is not performed at all,

2. it is performed incorrectly and the error is detectable, or

3. it is performed completely and correctly.

An "intentions list" records in non-volatile storage the actions necessary to carry out the write commands of a transaction. Following a crash, the recovery technique will examine all intentions lists, erasing completed ones and carrying out the actions specified in the rest. Each list is deleted when all of its actions are completed. Within such a framework, Lampson and Sturgis sketch a proof that three-state locks provide proper operation of a multiple-node system in the face of possible crashes and recoveries.

Three-state locking has also been proposed to handle mutual consistency among multiple copies of data, without deadlock. Peebles has pointed out, however [PEEB77], that while the solutions are deadlock-free for multiple copies of a single data item, they do not generalize to multiple-item updates whose write-sets overlap. For example, if transaction 1 required updating values of data items A, B, C, and D while transaction 2 required updating B, C, and E, the three-state lock solutions so far proposed are not adequate. This general case would require an accompanying scheme to deal with deadlock.

A.2.2 Consistency

The problem of consistency can be further divided into two parts: (1) checking the accuracy of every data item value entered into the database, and (2) maintaining that accuracy in the face of multiple, concurrent accesses involving updates which may be potentially contradictory or conflicting.

A.2.2.1 Accuracy.

Accuracy checking is usually handled from the point of view of plausibility rather than trying to check the value of each item in relation to all other items in the database. Such plausibility criteria can be established through consistency (sometimes called integrity) constraints which usually amount to specification of some set properties. For example, in a banking system, deposits should have values greater than zero and should be accepted only for accounts known to exist, and the day's total balance should equal the prior balance plus deposits less withdrawals. Cancelling errors, such as can occur in sums for example, will not be detectable in this way.

Consistency constraints are still probably the appropriate way to handle the accuracy problems in a DDBMS as well as in a centralized system. Some additional considerations involve: (1) needing to store constraints along with data, (2) keeping multiple copies of constraints

consistent as well as copies of data, and (3) handling partitioned data where constraints apply to data aggregates which span machines. There does not yet seem to be any literature which specifically addresses these issues.

A.2.2.2 Concurrent Access Control.

Concurrent access control is a problem whenever multiple users are allowed to work simultaneously on a database. Particularly when a database transaction can involve more than a single data item, it is important to prevent interfering accesses by simultaneous transactions involving updates. In a distributed database context, we can call this "internal" consistency to show it refers to non-redundant data within a single-copy database. For example, consider two database users, A and B, both accessing a payroll file. A's job is to give all employees in department 10 a 5% raise in salary. B's job is to give all employees with salaries less than $5000 a $500 raise. If C and D had salaries of $4800, the new salaries could be either $5040 (if A updated first) or $5300 (if B updated first). Concurrent updating by A and B could end up with C and D getting different salaries -- clearly an undesirable, and probably an incorrect result. Even if only C had been involved in the update, there is a question as to which result was actually intended for C. Internal consistency is usually handled by locking the data items to be updated so that concurrent accesses are prevented and

operations are strictly serialized. The order is usually established according to some predefined scheme based on mechanisms such as timestamping update requests or assigning priorities to all users. Such a scheme would have to represent a central or universally agreed-upon policy to be applied uniformly by each node.

In addition to internal consistency, a DDBMS must further provide "mutual" consistency among any copies of data partitions or an entire database. That is, if all updating activity stops, the copies must be identical after some reasonable transient time. In general, to provide mutual consistency in a distributed database, it has been assumed that application of any update must be synchronized among all data copies. That is, updates for the same data item must be ordered so that their application will not give different results in different copies.

The various approaches to concurrent access control have been stated in diverse ways and with widely differing terminologies. Performance comparisons of the algorithms are just starting to be done [GARC78] and formalisms for comparison are beginning to develop [GELE78 and BERN78]. LeLann has suggested [LELA78] that one way to group the research on solutions is according to the underlying mechanisms:

* physical time,

* logical time or timestamps,
* explicit control privilege,
* event counts or sequence identifiers.

These categories provide a convenient outline for our discussions here, and we will discuss one example for each.

A.2.2.2.1 Physical Time.

Lamport has shown that mutual exclusion in the use of distributed resources can be achieved using physical clocks which satisfy certain conditions (discussed in the next paragraph) [LAMP78]. The basis of this approach is that the sending and receiving of messages in a distributed system determine a unique partial ordering of the message events. For example, within a single node, the events are ordered by their times of occurrence, so that if event A occurred before event B we have the logical relation: A "happened before" B. Between nodes the ordering of events can be established only from physically related events such as A representing the sending of a particular message by one node and B representing the receipt of that same message by another node. In this case, the two events are also logically related by: A "happened before" B. Within the entire distributed system, "happened before" is only a partial ordering since there will be sets of events which are not logically related. The partial ordering can be extended to a total ordering by adding a rule to tell which of two unrelated (also called concurrent

if they happen between sets of related events) events came first. Any such total ordering effectively eliminates concurrent accesses; Lamport introduces physical clocks so that the rule and the resultant total ordering will give a serialization to the events that is appropriate as far as some (hypothetical) universal observer who sees the operation of the entire system is concerned.

The physical clocks are assumed to run at approximately the correct rate (i.e., different from the true rate only by some fraction kappa << 1), and to be sufficiently well synchronized (i.e., within some small epsilon of each other). For mu a quantity that is less than the shortest transmission time for interprocess (internode) messages, then if

epsilon / (1 - kappa) < or = mu

is true, anomalous behavior is impossible. Within these assumptions (the certain conditions mentioned above), the implementation rules for the clocks to provide the total ordering of events are: the clock rates are continuous between events, messages are timestamped by their senders, and message receivers reset their clocks to the sum of the timestamp and the minimum message delay (mu), if that sum is greater than their local clock time.

The main reasons for discussing this physical time scheme are that it is a formal treatment with proofs and that it lays the groundwork on which the next category of research, logical

time or timestamps, is built.

A.2.2.2.2 Logical Time.

Logical time, in the form of timestamps, was probably first introduced into distributed database systems by Johnson and Thomas [JOHN75]. Their timestamp consisted of a local time (from a physical clock) concatenated with a site identification, and was used to resolve the order of application of conflicting updates for multiple copies of data. One timestamp was said to precede another if the first time preceded the second, or if the times were equal and the first node ID was smaller than the second.

Thomas went on to use timestamps to completely coordinate multiple-copy distributed database updating in the majority consensus algorithm [THOM76]. In his scheme, each item in the database has associated with it a timestamp which represents the last time that item's value was updated. Update requests are assigned their own timestamps at origination and consist of a list of the items to be updated and a list of the base items (with timestamps) on which the update is predicated. The database manager in each node of the distributed system compares the base item timestamps in the request with the items in his own copy of the database. If any of the request timestamps are obsolete (i.e., older than the local copy timestamps), the update request is rejected (and may be saved

for later resubmission). If the timestamps are current and the request does not conflict with any others pending at that site, the vote is to accept and the request becomes pending at that site. A conflict occurs whenever the update items of one request overlap with the base items of another request. The conflicts are handled by assigning a priority order to the nodes of the network, and giving each update request the priority of its originating node. Requests conflicting with another of higher priority are rejected; requests conflicting with another of lower priority are set aside and voting is deferred. The originating node collects the votes on its update request; a majority consensus on acceptance allows update application in parallel throughout the system. Thomas showed that any rejection vote will ensure that a majority vote for acceptance cannot be achieved.

The major difference between Thomas's and Lamport's use of timestamps is in the use of the physical clocks. Lamport continually resets the clocks to maintain synchronization as closely as possible. Instead, Thomas prevents sequencing anomalies by having the update timestamp be assigned by the requesting node as the maximum value of either the local clock time or the latest timestamp of any of the base variables plus one. Thus the clocks can run at different rates, without any synchronization, and are only limited by the restriction that they never be set backwards.

A variety of other concurrent access control algorithms based on timestamps have been proposed [ROSE78, BADA78, GELE78]. One of these is SDD-1 [ROTH77], a DDBMS being designed and built by Computer Corporation of America, which is discussed in the section on Implementations.

A.2.2.2.3 Explicit Control Privilege.

A different approach to concurrency is based on control being explicitly assigned to a particular node. This would, in effect, be a DDBMS with distributed data and centralized control.

Alsberg and Day have suggested a fault-tolerant approach to explicit control, called n-host resiliency [ALSB76], where n hosts (n > 0, n < or = total number of hosts) must be aware of an update by receiving the request before either acknowledgement to the user or application to any local database copy. The point is that only one node is involved in conflict resolution or update sequencing (probably the simplest possible solution) and the goal is to provide reliable service (or be resilient) in the face of crashes (i.e., avoid dependence on a critical resource). To achieve this, the network hosts are divided into three groups:

1. users, which receive queries and updates from the external world,

2. a unique primary, and

3. back-ups, which are explicitly linearly ordered in some arbitrary prearranged manner.

Only the primary and back-ups have database copies. All updates are sent to the primary, who resolves any conflicts, applies the update to its local data copy and starts the request down the back-up chain. Each back-up acknowledges receipt of the update request from its predecessor and sends it on down the chain. The n-th node to apply the update (i.e., the n-1st back-up) returns the acknowledgement to the update originator, and the user can then proceed while propagation of the change through the back-up chain completes. The acknowledgement scheme supplies the n-host resiliency, by ensuring updates are never acknowledged to a user as complete until n hosts know about them. In this way, a crash of the primary or any other node does not cripple the system. Time-outs are used on acknowledgement expectations to detect any host failures. If a back-up fails, the chain is reestablished by linking the failed node's predecessor and successor. If this is not possible, a network partition occurs and consistency problems are unavoidable in the general case. If the primary fails, the back-ups will select a new primary. The point is that in general, operation is maintained despite failures unless fewer than n hosts remain.

A variation on this scheme has been proposed which allows broadcast of updates instead of linear propagation [GRAP76]. A similar scheme has been used by Stonebraker [STON76] for a distributed version of INGRES, which is discussed in the section on Implementations.

A.2.2.2.4 Sequence Identifiers.

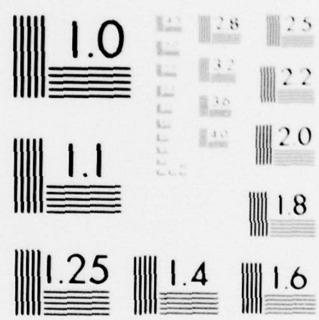
The fourth category of research into concurrency control uses event counts or sequence identifiers to implement distributed control over a distributed database without referring to either real clocks or timestamps.

LeLann has suggested [LELA78] explicitly sequencing update requests by use of tickets similar to ones used in a bakery to assign service order to the customers. The ticket order ensures that updates are processed in exactly the same order throughout the distributed system. To eliminate concurrency in accessing (as well as to avoid deadlock over data resources) and still have distributed control over the DDB, the database hosts of the system are arranged in a virtual ring by assigning permanent identification numbers in a sequential increasing fashion around the ring so that a predecessor and successor are defined for each node. A special message called the control token is circulated around the virtual ring to implement the ticketing facility. Tickets can be taken from the "dispenser" (the control token) only by

AD-A074 552 NAVAL UNDERWATER SYSTEMS CENTER NEW LONDON CT NEW LO--ETC F/G 5/2
PERFORMANCE AND TIMELINESS IN A DATABASE.(U)
JUL 79 L A DENOIA
UNCLASSIFIED NUSC-TR-6099 NL

3 OF 3
AD
A074552

END
DATE
FILMED
11-79
DDC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

the node owning the token and are assigned to pending update requests only after the control token has been accepted by that node's ring successor. Thus when an update request arrives at a particular node, it is assigned the next ticket that the node has available. If there are no tickets left, the request must wait in a pending queue until the control token arrives (back at that node) and the node can get more tickets.

Ticketed update requests are sent throughout the system. All nodes are restricted to applying updates in ticket order, so if preceding ticket numbers are missing, a newly arriving ticketed update must be queued until all lower ticket numbers have been processed. The token effectively acts in place of a centralized, master clock to provide global event sequencing.

LeLann discusses several performance variations on the basic scheme to allow increased parallelism within the system. One way involves preallocation of tickets for updates not yet pending, but expected before the control token returns. This requires that when the control token does return, any left-over, unused tickets must be flushed by sending out null updates. Otherwise one unused ticket could hold up all subsequently assigned ones indefinitely.

The particular advantage to the virtual ring with token and tickets as a concurrency solution is that there are no gaps in the ordering specified by the tickets the way there

are with timestamps. On receiving a ticketed update request, a node always knows how many missing updates will have to be received and processed before the current one, since it knows the ticket number of the last one it did process.

A.2.2.2.5 Conclusion

This section has presented a variety of solutions for the concurrent access control problem. The sample of schemes that has been presented is intended to point out the diversity of approach; analysis of assumptions and performance will be required to determine which solution is appropriate for a particular DDB system. Such analysis is just beginning to be done so that comparisons can be made [GARC78, BERN78].

A.3 SECURITY

The issue of security in distributed database management systems has not yet received a great deal of attention. The problems will almost certainly be at least as large as the sum of the problems of protecting the data stored within each node's piece of the database (internal security) and maintaining the security of communications between nodes. Internal security includes such problems as: verification of user identities, control of user access to data, control of user operations on data, and actual physical security of the computers and peripherals. It is becoming less clear that

even these are the same as for centralized systems. A particular example coming into focus as networks begin to develop general access mechanisms and routing schemes which involve multiple nodes is the difficulty involved in determining an individual user's location or network entry method. Even if he is asked, there is no way to validate his answer.

Encryption techniques will probably be applicable to communications security between nodes in the network, although some protection will be needed against possible sabotage by insertion of garbage streams into the communications subnet. If a user's location or entry method (e.g., local phone, long-distance phone, secured line) cannot be verified, however, encryption may be needed even internally.

Capability mechanisms [DENN76] provide one approach to centralized security which may be adaptable to distributed databases. Timestamps consisting of a node number appended to a local clock time can be used as unique (within an entire distributed system) identifiers if they are associated with data items upon entry into a database or upon value modifications (updates). Reed suggests [REED78] that updated values be considered as a series of timestamped versions of an original data item, and he keeps the relationship among the versions explicitly as a history associated with the original. Wyleczuk has proposed [WYLE79] a combination of capabilities

and versions to provide security and control access in a distributed environment. The use of versions connected by a history creates a context in which timestamps are naturally associated with every data item in the database. The timestamps can then be used both to identify particular items and to limit the effectiveness of the access privilege granted by a capability. Revocation of access privilege, for example, can be made automatic by including in a capability an expiration time for its effectiveness.

A.4 RELIABILITY

One important motivation for distributed database systems is the potential for improvements in availability and reliability that is created by having multiple copies of data and cooperating multiple sites of data control. Here, availability refers to the user's view of whether the system is functioning so that it can accomplish his job.
Reliability, on the other hand, begins with availability and adds concern for being able to continue operation even if some failure occurs. Centralized systems are often handicapped by failures in critical resources; a DDBMS could be more reliable by having few, if any, resources of a critical nature. For example, if one node becomes inoperable, another might be able to take over for it if the data required to handle the inoperable node's transactions are stored at both places. This is useful only insofar as the database integrity is

preserved when the failed node rejoins the distributed system. So back-up and recovery are as much a part of reliability as of integrity.

Multiple copies of data could also improve response time by providing quicker access to closer copies where communications delays would be shorter. Quicker response tends to make the system more available to (large numbers of) users by not requiring them to wait for the service they seek.

Increased reliability does not come for free, however. Keeping duplicates of any large amount of data will be costly in terms of memory and in terms of processing needed for consistency maintenance. This will have to be balanced against the costs of unreliability (delayed access or lost data). Tradeoffs will be necessary to determine how many copies of what information and programs are to be kept at what locations [MORG77]. Copies of data local to accessing programs can eliminate immediate communications delays but can cost update synchronization overhead. The amount of appropriate redundancy may well depend on the size of the database in a particular situation as well as on the patterns of data and program usage. Maintaining complete duplicate databases at each node will at least be too costly (in terms of storage space and restructuring time) for very large databases, those on the order of 10**12 bits [GERR77].

In general, updates to a database are considered to involve replacement or rewriting of the data items to be changed. More recently, M.E. Senko [SENK77] has suggested that a database should consist of all data values entered, with each data item consisting of a data value and a timestamp. Such a database fits very well with certain back-up techniques as journaling (recording all transactions against the database with identifying information) and with representation techniques as differential files (localization of database modifications in a storage area small relative to the entire database [SEVE76]). In this way updates are stored separately from the main archival file until there are enough updates to make their application to and the reorganization or restructuring of the storage for the main file be cost-effective.

The next two sections represent the state of the art in DDB research based specifically on reliability considerations.

A.4.1 Sunshine's Model

Sunshine has looked at developing an availability model for comparing the overall performance of a uniprocessor and several distributed processing system interconnection schemes [SUNS77]. His approach also supports the idea of multiple data copies. Availability is taken to be measurable by the probability that at least one processing element is

functioning so that it can accomplish work for a user and by the average amount of resources available to do the work. The cost of having redundancy among processors, programs, and data is included in the model.

The four configurations which were compared are: a single processor, N standby processors, N separate processors, and N interconnected processors. To make the total processing power the same for all candidates, each processing element in the systems of N processors is assumed to have 1/N times the processing power of the uniprocessor system. The results show that the amount of time the system is available can be improved over the uniprocessor case by using N standby or interconnected processors. Of these latter two, the interconnected scheme provides the greater average processing capacity (although both are less than the uniprocessor). Thus the redundancy required to achieve greater available time costs processing capacity. Further study is needed to examine the tradeoffs involved in different interconnection schemes.

The model is simple since it is a first attempt to formalize measures for reliability.

A.4.2 The National Software Works

One complete system which has been designed from the point of view of reliability is the National Software Works (NSW), described in [SHAP77]. The goal of NSW is to provide

through the ARPAnet a set of software development tools and a host-independent user interface to them. The monitor portion of NSW, the Works Manager, requires a synchronized, duplicate-copy distributed database to hold its access control, accounting, and auditing information, and its file catalog. The design of NSW is based on the ideas that a system is reliable if:

1. it is available for use,
2. all actions reported to a user as complete are correctly reflected in any future state of the system, and
3. a user can probe the system for the status of any action he has initiated and thus find out just what the state of the DB is at any time.

It is considered very unlikely that concurrent access conflicts will be a problem in NSW. The nature of the database precludes conflicts over anything other than: login, semaphore setting, or entry of new file names by users sharing name spaces. Nevertheless, mechanisms have been included to ensure reliable operation. First, each copy of the Works Manager explicitly assigns sequence numbers to all messages sent to other copies (M numbers). This ensures (1) proper ordering of the messages at the receivers and (2) detection and identification of lost messages, so that communications are reliable.

To handle updates, each data item has associated with it the number of times its value has been changed (R numbers). To request an update, the Works Manager gives its current R number for the data item and assigns a timestamp to the request. When the request arrives at another host, its R number is checked against that for the local copy of the data. If the incoming R is smaller, the request is discarded; if it is equal or larger, the local pending queues of all updates (there is one for each other Works Manager) are searched until (for each) either: (1) an update for that item with the same R and an earlier timestamp or (2) a message with a later timestamp is found. If all host queues satisfy (2), the incoming update can be applied. If any queues produce (1), the update with the earliest timestamp is selected for application first. It can be shown [SHAP77] that the detailed operation of these mechanisms produces correct and consistent results across the system.

A.5 DATA INCOMPATIBILITY

Problems with data compatibility will come up whenever different processors in a heterogeneous distributed system attempt to transfer data, since there are no computer industry standards for internal computer representation of data. Different processors use different numbers of bits to make up bytes and words, and the translation between corresponding data type formats may result in loss of precision. There are

different representations for integers (sign-magnitude, one's complement, two's complement), for floating point numbers (bit location of mantissa and exponent, number of bits for each, type of exponent representation such as excess-three, and even the base can be 2, 8 or 16) and so forth. Different treatments of round-off or truncation may give different results for numeric computations at different processing sites. Even where standard character sets are adhered to (ASCII and EBCDIC, for example), different uses of control characters or sequences are still common and must be translated. Composite data structures such as arrays or complex numbers are stored differently by various processors, and the possibility for embedded pointers complicates things even further.

Current approaches to the data incompatibility problem include: homogeneous networks (i.e., ignore the problem); negotiation of options as under ARPANET's Telnet or File Transfer Protocol; translators such as ARPA's Data Reconfiguration Service, for users of particular application programs; and translating to and from some pre-specified network standard, such as the proposed Intermediate Data Format [LEVI77]. Notice that these are all actually political strategies for circumventing the problems, and are not really technical solutions.

Levine's discussion [LEVI77] emphasizes that descriptive information about data must at least accompany it, and should really be considered part of it. This includes:

--definition or class (e.g., character, integer, instruction),

--internal format representation (e.g., two's complement),

--precision.

This issue is still a large problem, and there is little work reported in the literature.

A.6 ORGANIZATION

Research on the organization of distributed database systems has focused mainly on where to physically locate files and directories among the network nodes. Several cost models have been developed [e.g., CHUW73, LEVI76, CHAN76, MAHM76] and optimizations were found to be very expensive without development of suitable heuristics. Other organization problems include: file decomposition when partitioning is desired, dynamic movement of files as user access patterns change, file location to achieve some system goal such as maximum acceptable response time. Most of these are related to the cost/performance goals set for the database system by its designers and installers, and will be solved only on the basis of more experience with the performance of experimental systems.

A.7 IMPLEMENTATIONS

No full-blown general-purpose distributed database management systems are advertised as existing today. The purpose of this section, then, is to describe briefly some of the planned systems and experimental implementations which will actually lead to DDBMSs that work. It is difficult to determine whether any of these systems will allow performance evaluation of various implementation alternatives, such as comparison of different concurrent access control methods. Notice that research results from inside the computer industry are conspicuously absent, probably for company proprietary reasons.

A.7.1 INGRES

INGRES is a database management system for relational databases [STON76]. It is being implemented for PDP-11 computers on top of the UNIX operating system. The extensions of INGRES being made to handle a distributed database [STON77] assume (1) a network of machines homogeneous in that they all run UNIX, and (2) a UNIX to UNIX communication capability.

Distributed INGRES is a primary, back-up scheme based on Alsberg's n-host resiliency approach, except that different data objects can have different primary sites. A particular goal of INGRES [STON78] is to provide a unified distributed

solution to both the concurrency control and crash recovery aspects of the integrity problem. It is assumed that the system exhibits a high degree of locality of reference; i.e., that the data required to process most transactions (about 95% is suggested) are stored locally. Consequently, performance is based specifically on whether data to be referenced is local or not; there is no consideration of "nearby" or "close neighbor" sites.

Two approaches are provided for concurrent access control and for crash recovery:

* the "performance" oriented approach provides the fastest possible response time, and may have consistency problems if the system is not crash-free; and
* the "reliability" oriented approach ensures mutual consistency among copies, and may pay a large performance penalty.

INGRES updates are always directed to the primary copies, but the high performance approach will allow users to specify that they want retrievals to be answered quickly from the local data copy without regard for what else is occurring in the system.

Concurrency control is handled by having a local concurrency controller (cc) at each site, running some appropriate algorithms. Deadlock is a particular problem since there are multiple primary sites. The cc is responsible

for detecting any local deadlocks within its own machine, and backing out or undoing the effects of an appropriate local transaction. In addition, cc must report any wait-for conditions caused by conflicting lock requests to a program called SNOOP. SNOOP is located at some particular site just to construct global wait-for graphs so that inter-site deadlock can be detected and handled. In other words, SNOOP is the primary site for the wait-for data. Such a centralized deadlock control solution is not only simple, it is considered by Stonebraker particularly appropriate for the high degree of locality of reference on which INGRES is predicated. Details of the performance and reliability algorithms are described in [STON78].

The organization of INGRES [STON77] is based on n-ary relations [MART76], which may be distributed across machines in subsets of the rows of the relation. Two types of relations are supported: regular relations, which are shared by the network, and local relations, which are visible only to local users on the machine where the relation is stored. Each DDB site keeps a complete system catalog for all its locally resident relations, and keeps a complete list of names of all the regular relations in the network. When access is made to a non-resident relation, the appropriate catalog entry is requested and saved. Such information is not updated and is simply discarded after some pre-specified amount of time.

For retrieval strategy, INGRES uses decomposition of a transaction dealing with multiple data items into a sequence of transactions which deal only with single data items and some manipulations to represent the appropriate combinations or restrictions of information required to produce the final result. The centralized strategy is extended to handle the distributed case by moving parts of data sets and by combining information among nodes, as appropriate [STON77].

INGRES is a particular combination of distributed and centralized control; performance evaluation of the implementation at Berkeley is bound to give interesting results. How generally comparable the performance data will be to other implementations remains to be seen.

A.7.2 SDD-1

Computer Corporation of America is designing and implementing a logical time (timestamped) distributed database system called SDD-1 [ROTH77, CCA 78]. A particular goal of SDD-1 is to improve performance compared to a multiple-copy system with a complicated synchronization mechanism. To achieve this, transaction class definitions are developed and analyzed at DDB design time to determine what degree of synchronization is required for each class. In this way, transactions which are known to be non-conflicting can run very simple serializing protocols. Because class definition

and analysis is done ahead of time, run-time coordination uses fast table look-up to choose protocols approppriate for whatever transactions occur. Only transactions which might conflict or unforeseen types of transactions will be required to run the full synchronization protocols with their higher overhead.

The organization of SDD-1 is based on a relational data model. Each relation to be distributed across sites for physical storage is partitioned into rectangular subrelations called fragments as the unit of distribution. Redundancy is possible at the fragment level; it is not expected that copies of the entire database will be realistic. A collection of fragments which make up a complete, non-redundant copy of the database is called a materialization. Directories are treated in the same way as ordinary user data, except that directory information for directories themselves is stored at every data module. Timestamps are associated with each physically stored copy of a data item and represent the time at which the last transaction modified that copy of the data. Since data items are the smallest pieces of information that can be independently accessed in the DDB (somewhat in analogy to the fields of a logical record), considerable extra storage space will be required for the timestamped items as compared with the space for just the items.

SDD-1 uses an optimization technique to break distributed transactions into a combination of local processing on data located at a single site and of moving data among sites so that it can be processed. The object is to minimize the total cost. The process basically consists of iteratively trying to improve the cost of a simply chosen strategy by examining each step to see if it can be replaced by a lower cost combination of other moves and local processing. Moving cost is assumed proportional to the amount of data moved; processing cost depends on the operation and the size of the relation.

Appendix B

DETAILS OF THE MODEL ANALYSIS

B.1 THE BASIC M/M/1 QUEUING MODEL

The notation M/M/1 specifies first that the interarrival times are exponentially distributed (M), next that the distribution of service times is exponential (M), and finally that there is just a single server (1) [KLEI75]. The exponential distributions are characterized by their means; we will use L to represent the average arrival rate to the system, and XBAR for the average service time (we could have used an average service rate such as mu = 1 / XBAR, but XBAR is more convenient).

For any single server system the "utilization" is defined to be the product of the average arrival rate and the average service time,

rho = L * XBAR ;

so that the utilization represents the ratio of the rate at which work enters the system to the rate at which work is performed. Utilization is also interpreted as the fraction of the time that the server is actually busy performing work.

The measure of queuing system behavior with which we are particularly concerned in this analysis is the average total time, T, spent in the system by a transaction (both waiting in the queue and being served). For M/M/1, the average total time spent in the system is

T = XBAR / (1 - rho) .

B.1.1 Centralized Management

In the centralized system, we have two types of arrivals: updates and retrievals. If we call the update arrival rate from each terminal LU, and the retrieval arrival rate from each terminal LR, then for our central system which has n terminals per node and only N = 1 (i.e., the central) node, the total arrival rate to the server is

L = n * N * LR + n * N * LU .

Rather than simplify the expression by substituting in the N = 1, we will keep it in this form for easier comparability with the expressions for distributed systems.

In general, updates and retrievals may require different amounts of service time. We will call the average service time required for an update XU, and the average service time required for a retrieval XR. Then the average service time for the entire system (XBAR) depends both on the individual times for retrievals and updates (XR, XU) and on the relative frequencies of occurrence:

$$XBAR = \frac{n*N*LR}{L} * XR + \frac{n*N*LU}{L} * XU \quad .$$

For A the ratio of update to retrieval arrivals (i.e., A = LU / LR), we rewrite

$$L = n * N * (LU + LR) = n * N * LR * (A+1) \quad \text{and}$$

$$XBAR = \frac{A * XU + XR}{A+1} \quad .$$

This gives us

$$rho = n * N * LR * (A * XU + XR)$$

and we see that

$$T = \frac{A * XU + XR}{(A + 1)*(1 - rho)} \quad .$$

We are, of course, restricted on the average to rho < 1 in order that the system be stable and not bog down.

Each transaction in the centralized system must first be sent to the central node, then enter the queue, be serviced, and finally be sent back to the terminal. The average response time for a transaction is thus

$$RT = CD1 + T + CD2 \ ,$$

where CD1 is the communications delay from terminal to central node and CD2 is the communications delay from the central node back to the terminal. In order to focus on the issues of the management schemes being compared, we will simplify the detail of the communications contributions by taking CD1 = CD2 = CD, a constant. In our first order treatment, this could be considered as a distribution of communication times

represented by its average value.

Substituting for T and combining the CD terms, we get

$$RT = 2*CD + \frac{A * XU + XR}{(A + 1)*(1 - rho)} .$$

B.1.2 Master/Slave Analysis

In the operation of a master/slave system, each slave node is responsible for processing the retrieval transactions it receives from its own locally connected terminals and for applying all of the system update transactions against its local (complete duplicate) copy of the database. This gives a total average arrival rate at a slave node of

$$Ls = n * LR + n * N * LU .$$

Using the proper proportions of each transaction type, we see that the average service time within a slave node is

$$XBARs = \frac{n * LR}{Ls} * XR + \frac{n * N * LU}{Ls} * XU .$$

Substituting A = LU / LR and simplifying, we get

$$Ls = n * LR * (N * A + 1) \quad \text{and}$$

$$XBARs = \frac{N * A * XU + XR}{N * A + 1} .$$

The master node is responsible for all of the system update processing (such as sequencing, resolving conflicts, application) before the accepted update is sent back out to the slaves for local application to the slave database copies.

Since the master also has locally connected terminals for which it handles retrievals, the total average transaction arrival rate at the master is

$$Lm = n * N * LU + n * LR ,$$

which we immediately recognize as the same as Ls for the slave nodes. The same kinds and rates of transaction arrivals apply also to the back-up node (which provides the 2-host resiliency) in the master/slave system. Since all nodes have now been shown to handle the same total average arrival rates with the same average service times, we can say that in a master/slave system, for each node,

$$L = n * LR * (N * A + 1) \quad \text{and}$$

$$XBAR = \frac{N * A * XU + XR}{N * A + 1} .$$

From L and XBAR, we get that the utilization of each node is

$$rho = n * LR * (N * A * XU + XR)$$

and that the average time spent in any one node is

$$T = \frac{N * A * XU + XR}{(N * A + 1)*(1 - rho)} .$$

The average response time throughout the system must take into account both different types of transactions. Using the notation p(event) to represent the probability of that event, we have

$$RT = p(update) * RTU + p(retrieval) * RTR ,$$

where RTU and RTR are the average response times for updates and retrievals, respectively. Now considering the different

node types and using p(type) to represent the probability that the node in question is that type, we extend the previous equation to

RT = p(master) * p(update) * [illegible]

+ p(back-up) * p(update) * [illegible]

+ p(slave) * p(update) * [illegible]

+ p(retrieval) * T .

The terms within the brackets represent transmission to master, update application by master, transmission to back-up, application by back-up, and transmission of acknowledgement to the user, respectively. Collecting like terms in T, we get

RT = p(master) * p(update) * 2 * CD

+ p(back-up) * p(update) * 2 * CD

+ p(slave) * p(update) * 3 * CD

+ [2*p(update) + p(retrieval)] * T .

For one master, one back-up, N-2 slaves, and an update/retrieval ratio of A,

RT = (1/N) * [A/(A+1)] * 2 * CD

+ (1/N) * [A/(A+1)] * 2 * CD

+ [(N-2)/N] * [A/(A+1)] * 3 * CD

+ [2*A/(A+1) + 1/(A+1)] * T ,

which simplifies to

$$RT = \frac{A}{A+1} * \frac{3N-2}{N} * CD + \frac{2A+1}{A+1} * \frac{N*A*XU + XR}{(N*A+1)*(1-rho)} .$$

In the master/slave system, since retrievals are all handled locally, they contribute no network messages to the cost predictor. The contribution of the updates is weighted by their probability of occurrence, so that the system cost in terms of the average number of messages required to process any transaction is

$$NM = \frac{A}{A+1} * \frac{3*N-2}{N} \quad .$$

B.1.3 Synchronized Management

The control token, as any other message or transaction, requires time CD to be passed from one node to its successor. The length of time for a circuit of N nodes, i.e., the time between successive token arrivals at a single node, is then N * CD. During that time, the average number of arrivals at each node is n * N * CD * LR retrievals and n * N * CD * LU updates. All of the updates from all of the nodes go into the batch and the retrievals are saved locally at each node to be added at the end of the batch processing, for a total batch size at any node of

$$N * (n * N * CD * LU) + (n * N * CD * LR)$$
$$= n * N * CD * (N * LU + LR).$$

The time required to process the batch sequentially is

$$TBX = n * N * CD * (N * LU * XU + LR * XR).$$

Since the time between batches is N * CD, we can define the utilization of the server at each node to be

rho = TBX / (N * CD),

which is (as usual) the fraction of time the server is busy. We are constrained, as mentioned before, to have

rho = n * (N * LU * XU + LR * XR) < 1.

The average response time for a transaction will be made up of three parts:

* waiting for the control token,
* waiting for the predecessors in the queue to be served, and
* actual service time.

Using our assumption of activity being uniformly distributed across all terminals and nodes of the network, we will say that the average wait for the control token is half of the time for a virtual circuit, .5 * N * CD. The order in the queue is updates from all other nodes, local updates and then local retrievals. All local transactions wait for non-local updates (from the other N-1 nodes) to be processed,

n * (N-1) * CD * N * A * LR * XU

(recalling that LU = A * LR). Since average service times are constant for a given type of transaction, the uniform distribution says the average wait for local updates will be for half the total number and the average wait for local retrievals will be for all updates plus half the number of retrievals in the batch. Thus we have

$$RT = .5 * N * CD + n * (N-1) * N * CD * A * LR * XU$$

$$+ p(update) * (.5 * n * N * CD * A * LR * XU)$$

$$+ p(retrieval) * (n*N*CD*A*LR*XU + .5*n*N*CD*LR*XR) \quad .$$

Putting in the probabilities and combining waits with service times gives

$$RT = .5*N*CD + n*(N-1)*N*CD*A*LR*XU$$

$$+ [A/(A+1)] * (.5*n*N*CD*A*LR*XU + XU)$$

$$+ [1/(A+1)] * [n*N*CD*LR*(A*XU+.5*XR) + XR] \quad .$$

If we treat the control token as having the updates to be circulated appended to it, the combined token/update package can be considered as a message. For each update, then, N messages are required to complete the circulation, and the number of messages averaged over both updates and retrievals gives us

$$NM = [A/(A+1)] * N \; .$$

B.2 EXTENSION OF THE MODEL TO HANDLE QR/BI RETRIEVALS

From Kleinrock's [KLEI76] derivations for non-preemptive head-of-the-line priority queuing, we know that the average delay (WO) due to a customer already in service as observed by a new arrival is

$$WO = \sum_{i=1}^{P} (Li * XiSQBAR / 2)$$

where Li is the average arrival rate for the i-th priority class, XiSQBAR is the second moment of the service time distribution for the i-th priority class, and there are P priority classes in the queue. We can simplify this expression by taking advantage of our knowledge of the exponential distribution where the second moment is just twice the square of the first moment [KLEI75]:

$$\text{W0} = \sum_{i=1}^{P} \text{Li * XiBAR**2}$$

where now XiBAR is the first moment of the service distribution for the i-th priority class, and that is just what we have been working with, the average service time! For the QR-BI model, P = 2; the QR transactions will be one priority class and the BI and update transactions will be the other. So for QR-BI we will work with just

$$\text{W0 = L1 * X1BAR**2 + L2 * X2BAR**2} .$$

For each of the management schemes, then, we will have to discover the average arrival rates and average service times by priority class.

Each new arrival to the priority queuing system will have to wait, not only for any customer in service when he arrives, but also for any higher priority customers who are already in the queue or who arrive before our particular arrival gets into service. Kleinrock shows [KLEI76] that these waits sum together to give the average wait time in queue for customers

of each priority class,

$$W_p = W_0 + \sum_{i=p}^{P} X_iBAR \cdot L_i \cdot W_i + \sum_{i=p+1}^{P} X_iBAR \cdot L_i \cdot W_p$$

for p = 1, 2,...,P . Notice that this set of equations is entirely based on just the things that we have assumed across the database management system models.

The solutions to the wait time equations are more conveniently written in terms of a quantity sigma, which Kleinrock defines for each priority class p as:

$$\text{sigma}(p) = \sum_{i=p}^{P} X_iBAR \cdot L_i = \sum_{i=p}^{P} \text{rho}(i) \quad .$$

Then the solutions to the waiting times become

$$W_p = \frac{W_0}{[1-\text{sigma}(p)] \cdot [1-\text{sigma}(p+1)]}$$

for p = 1, 2, ..., P .

We restrict the solution to the two priority classes for the QR-BI modeling:

p = 1 for updates and BI retrievals and

p = 2 for QR retrievals

so that the variables in the equations will be L1 and X1BAR, L2 and X2BAR, sigma1 = sigma(1) and sigma2 = sigma(2), W1 and W2. The average wait due to some transaction already in service becomes

WO = L1 * (X1BAR)**2 + L2 * (X2BAR)**2 ,

and the average wait times (in queue) will be

W1 = WO / [(1-sigma1) * (1-sigma2)] and

W2 = WO / (1-sigma2) .

Since the average total time (in queue and in service) is the sum of the average wait and average service times, we get

TU = W1 + XUBAR for updates,

TBI = W1 + XRBAR for BI retrievals, and

TQR = W2 + XRBAR for QR retrievals.

These processing times will be combined with various communications delays according to the management schemes to yield average system response times. As usual, solutions will be constrained by utilizations less than one:

rho = L1 * X1BAR + L2 * X2BAR < 1 .

B.2.1 Centralized Management

In the centralized database management scheme, all transactions are handled at the central node. So for n terminals, all communicating with the (N=1, single) central node, the arrival rates within priority classes are

L1 = n * N * (LU + LBI) and

L2 = n * N * LQR ,

where LU is the update arrival rate per terminal (as before), LBI is the arrival rate per terminal for BI retrievals, and LQR is the arrival rate per terminal for QR retrievals.

Since we have grouped the BI retrievals in the same priority class as the updates and the two may have different average service times (XR and XU), we must use a mean value for the class average:

$$X1BAR = \frac{LU}{LU+LBI} * XU + \frac{LBI}{LU+LBI} * XR \quad .$$

In contrast, the high priority class has only QR retrievals to service, and we have simply

$$X2BAR = XR \; .$$

Noticing that the total retrieval arrival rate per terminal is LR = LQR + LBI, we again define A = LU / LR as the ratio of update to retrieval transactions. We choose for another parameter the ratio of the QR to the BI requests, B = LQR / LBI. Rewriting in these terms the basic quantities so far, we get

$$L1 = n * N * LBI * [\ A*(B+1) + 1\ ]$$

$$X1BAR = \frac{A * (B+1) * XU + XR}{A * (B+1) + 1}$$

$$L2 = n * N * B * LBI$$

$$X2BAR = XR \; .$$

From these expressions we calculate

$$W0 = \frac{n*N*LBI*\ ([A*(B+1)*XU + XR]**2)}{A*(B+1) + 1} + n*N*B*LBI*\ (XR**2) \; ,$$

sigma1 = n * N * LBI * [A*(B+1)*XU + (B+1)*XR] , and

sigma2 = n * N * B * LBI * XR .

The wait times for each priority class are thus

W1 = W0 / [(1-sigma1)*(1-sigma2)] and

W2 = W0 / (1-sigma2) .

We combine the waiting and service times to get the total time required in the system for the different transaction types:

TU = W1 + XU for updates,

TBI = W1 + XR for BI retrievals, and

TQR = W2 + XR for the QR retrievals.

The utilization restriction is that

1 > rho = rho(1) + rho(2) = L1*X1BAR + L2*X2BAR .

As in the basic case of centralized management, the average response time for the entire system is made up of both the communication delays and the average time spent in the central node

$$RT = 2*CD + \frac{A}{A+1}*TU + \frac{1}{(A+1)*(B+1)}*TBI + \frac{B}{(A+1)*(B+1)}*TQR \quad .$$

The only messages crossing the network for the QR-BI centralized system are the same communications between the terminals and the central node as in the basic case, so

NM = 2

for our cost prediction.

B.2.2 Master/Slave Management Analysis

Let us begin by considering the master node, since under the QR-BI scheme the master picks up the additional work of answering all the BI retrievals for the network. This gives a low-priority retrieval arrival rate of

$$L1 = n*N*(LU + LBI) = n*N*LBI*[A*(B+1) + 1]$$

and an average service time of

$$X1BAR = \frac{A * (B+1) * XU + XR}{A * (B+1) + 1} .$$

The high-priority arrival rate at the master is just its share of the QR requests as input from its locally connected terminals

$$L2 = n * LQR = n * B * LBI ,$$

and the average service time is

$$X2BAR = XR .$$

We now have enough information for

$$W0m = \frac{n*N*LBI* ([A*(B+1)*XU + XR]**2)}{A*(B+1) + 1} + n*B*LBI* (XR**2) ,$$

$$sigma1m = n * LBI * [N*A*(B+1)*XU + (N+B)*XR] , \text{ and}$$

$$sigma2m = n * B * LBI * XR .$$

As usual, the total time spent in the system by each transaction type is the sum of its wait and service times:

TUm = W1m + XU for updates,

TBIm = W1m + XR for BIs, and

TQRm = W2m + XR for QRs.

The slave and back-up nodes have exactly the same arrivals for

QR-BI as they did for the basic case. The updates are processed in the low-priority class, for which

L1s = n * N * LU = n * N * A * (B+1) * LBI and

X1BARs = XU .

The behavior of the high-priority class depends only on the QR retrievals, so

L2s = n * LQR = n * B * LBI and

X2BARs = XR .

Calculating our now-familiar quantities

WOs = n*N*LBI*A*(B+1)*(XU**2) + n*B*LBI*(XR**2) ,

sigma1s = n*LBI* [N*A*(B+1)*XU + B*XR] , and

sigma2s = n*B*LBI*XR ,

we are prepared to get the average total time in the system by transaction type:

TUs = W1s + XU for updates at the slaves,

TQRs = W2s + XR for QRs at the slaves, and

TUb = TUs for updates at the back-up, and

TQRb = TQRs for QRs at the back-up.

As in the basic case, updates are handled by the master and back-up (with 2-host resiliency),

RTU = [(3N-2)/N] * CD + TUm + TUb .

All BI's which arrive at nodes other than the master are sent there for processing, while the master's BIs are local. The average response over the system for a BI retrieval is thus

$$RTBI = \frac{N-1}{N} * (2*CD + TBIm) + \frac{1}{N} * TBIm \quad .$$

QR retrievals are all processed at their local nodes, so that on the average

$$RTQR = \frac{1}{N} * TQRm + \frac{1}{N} * TQRb + \frac{N-2}{N} * TQRs \quad .$$

We can combine transaction response times with the frequencies of occurrence for the average system response

$$RT = \frac{A}{A+1} * \frac{3N-2}{N} * CD + TUm + TUb$$

$$+ \frac{1}{(A+1)*(B+1)} * \frac{N-1}{N} * 2 * CD + TBIm$$

$$+ \frac{B}{(A+1)*(B+1)} * \frac{TQRm}{N} + \frac{TQRb}{N} + TQRs*\frac{N-2}{N} \quad .$$

The cost prediction for the QR-BI model of master/slave management simply adds the contribution for the BIs to the update contribution of the basic case:

$$NM = \frac{A}{A+1} * \frac{3N-2}{N} + \frac{1}{(A+1)*(B+1)} * \frac{N-1}{N} * 2 \quad .$$

B.2.3 Analysis of Synchronized Management

Unlike the previous QR-BI models, where we have been able to capitalize directly on Kleinrock's presentation of head-of-the-line priority queuing, the batch nature of sending transactions around the virtual circuit with the control token destroys the premise of the wait time analysis. The arrivals are no longer Poisson to each priority class. We will attempt

to cover the situation by breaking our consideration of the system activity into two parts: batch processing, during which QR retrievals are given non-preemptive priority over batch items; and processing between batches, when QRs arrive and wait for service only with any previous QRs.

In general, the k-th local update in the batch must wait while all the updates in the batch from the other nodes are processed, while its k-1 local predecessor updates are served, and for any local QR transactions which arrive before the k-th update is itself accepted for processing. We can express this by

$$WU(k) = (N-1)*(n*N*CD*LU)*XU + (k-1)*XU + LQR*WU(k)*XR ,$$

where WU(k) stands for the wait time experienced by the k-th local update, and where the last term is the QR arrival rate times the interval of interest times the required service time. Solving this equation for the wait time itself, we find

$$WU(k) = \frac{(N-1)*n*N*CD*LU*XU + (k-1)*XU}{1 - LQR*XR} .$$

All of the updates are located together in the batch, so that, on the average, the uniform distribution says the wait time will be for half of the total number of local updates actually in the batch:

$$WU = \frac{(N-1)*n*N*CD*LU*XU + (1/2)*(n*N*CD*LU*XU-1)}{1 - LQR*XR} .$$

Similarly, the local BI retrievals in the batch must wait for all of the updates to be served, for half (on the average) of the BIs themselves, and for any arriving QRs. In a manner directly analogous to the development of WU, then, we get

$$WBI = \frac{N*n*N*CD*LU*XU + (1/2)*n*N*CD*LBI*XR}{1 - LQR*XR} .$$

To express the average wait time for the QR retrievals, we must take into account the difference between arrivals which arise during and between batch processing. Let us define, as in the basic case, the average utilization of the server due to the batch processing:

$$rho(b) = TBX / (N*CD),$$

where TBX is the time to process the items in the batch sequentially and N*CD is the total time required for the control token to complete a circuit of the virtual ring. We can see that

$$TBX = N*n*N*CD*LU*XU + n*N*CD*LBI*XR ,$$

giving us

$$\begin{aligned} rho(b) &= n*N*LU*XU + n*LBI*XR \\ &= n*LBI* [\ N*(A+1)*B*XU + XR\ ] . \end{aligned}$$

So, on the average, QR transactions will arrive at the service queue to find batch processing in progress a fraction rho(b) of the time. Since we have chosen a non-preemptive scheme, the QRs will have to wait an extra amount of time for the batch item in progress to be finished. WO, the wait due to

some customer being in service when our new QR arrives, is just

$$WO = rho(b) * \text{average time left in service} .$$

From the memoryless property of the exponential distribution [KLEI75], we get that the average time remaining in service is just the average service time for the batch items,

$$\frac{n*N*N*CD*LU*XU + n*N*CD*LBI*XR}{n*N*N*CD*LU + n*N*CD*LBI} .$$

Substituting this back gives us

$$WO = rho(b) * \frac{N * LU * XU + LBI * XR}{N * LU + LBI} .$$

In addition to WO, the QR transactions may have to wait for any preceding QR arrivals to be processed, giving us a total average time in queue of

$$WQR = = WO + \frac{n * LQR * (XR**2)}{1 - n * LQR * XR} .$$

In looking to describe the response time, we realize that the BIs and updates must wait for the control token to arrive before batch processing can even take place. Since communication delays between successors in the virtual ring have been considered constant, the uniform distribution tells us the delay waiting for the token will be half a virtual circuit on the average. This makes transaction response times

$RTU = (N/2) * CD + WU + XU$,

$RTBI = (N/2) * CD + WBI + XR$, and

$RTQR = WQR + XR$.

Weighting these individual times by the probability of each transaction type gives an average system response time of

$$RT = \frac{A}{A+1} * RTU + \frac{1}{(A+1)*(B+1)} * RTBI + \frac{B}{(A+1)*(B+1)} * RTQR \quad .$$

As in the basic case for synchronized management, it is only the updates which contribute network message traffic to the cost prediction,

$$NM = [\ A/(A+1)\ ] * N \ .$$

B.2.4 Analysis of Delayed Synchronization Management

Under the delayed synchronization scheme, the low-priority arrival rate is

$L1 = n * LBI * [\ A*(B+1) + N\ ]$, since each node handles its own updates, while the BI retrievals complete a circuit of a virtual ring in order to find the most recent copies of data. The corresponding average service time for the class is

$$X1BAR = \frac{A*(B+1)*XU + N*XR}{A*(B+1) + N} \quad .$$

If we add up the QR arrivals which can be processed at a given node, including local requests being processed locally and requests transferred from other nodes, the aggregate arrival rate is

$$L2 = n * LQR = n * B * LBI$$

and the average service time for the QR priority class is

$$X2BAR = XR \ .$$

From these quantities, we calculate

$$W0 = \frac{n\ast LBI\ast\ ([A\ast(B+1)\ast XU + N\ast XR]\ast\ast 2)}{A\ast(B+1) + N} + n\ast B\ast LBI\ast\ (XR\ast\ast 2)$$

along with

$$\text{sigma1} = n \ast LBI \ast [\ A \ast (B+1) \ast XU + (N+B) \ast XR\ ]$$ and

$$\text{sigma2} = n \ast B \ast LBI \ast XR\ .$$

Using the now familiar expressions for total time by transaction type,

$$TU\ = W1 + XU\ ,$$

$$TBI = W1 + XR\ ,$$ and

$$TQR = W2 + XR,$$

and the frequencies of occurrence, we get

$$RT = \frac{A}{A+1} \ast TU + \frac{1}{(A+1)\ast(B+1)} \ast N \ast [\ CD + TBI\ ]$$

$$+ \frac{B}{(A+1)\ast(B+1)} \ast (\ [p(\text{local})]\ast TQR + [1-p(\text{local})]\ast[2\ast CD+TQR]\ )$$

in a manner analogous to the basic case.

Network messages are required to send the BIs around the virtual circuit and to pass QRs which cannot be answered locally to the nearest neighbor. Averaging over all the types of transactions, then, we get a cost prediction of

$$NM = \frac{1}{(A+1)\ast(B+1)} \ast N + \frac{B}{(A+1)\ast(B+1)} \ast [1-p(\text{local})] \ast 2\ ,$$

with terms in powers of p(local) neglected from our locality of reference assumption.

INITIAL DISTRIBUTION LIST

Addressee	No. of Copies
OASN (Dr. R. Hoglund)	1
ONR, ONR-200	1
CNO, OP-098	1
NRL (Y.S. Wu)	1
NAVSEASYSCOM, SEA-003, -06V	2
NAVAIRDEVCEN	1
NOSC (D. Eddington)	1
NAVSURFWPNCEN (J. Miller)	1
NAVPGSCOL (Comp. Sci. Dept.)	1
DDC	12
ENGINEERING SOCIETIES LIBRARY, UNITED ENGRING CTR	1